VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

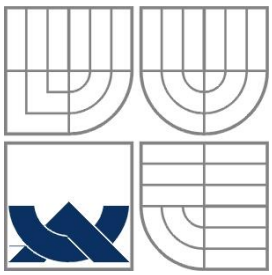
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

GRAFICKÉ ROZHRANIE PRE DOSKOVÉ HRY NA PLATFORME ANDROID

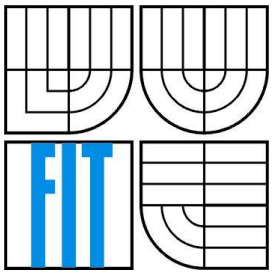
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE MARTIN KÁČERIK
AUTHOR

BRNO 2014



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

GRAFICKÉ ROZHRANIE PRE DOSKOVÉ HRY NA PLATFORME ANDROID

GRAPHICAL INTERFACE FOR BOARD GAMES ON ANDROID PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
MARTIN KÁČERIK

VEDOUCÍ PRÁCE
SUPERVISOR
ING. LUKÁŠ POLOK

BRNO 2014

Abstrakt

Tato bakalářská práce se zabývá zobrazením třírozměrné scény deskové hry. Popisuje možnosti zobrazování třírozměrné grafiky na platformě Android. Uvádí principy vykreslování pokročilými zobrazovacími metodami. Výsledným produktem je aplikace pro Android, sloužící jako správce modulů obsahujících implementaci deskových her. Hry jsou zobrazovány pomocí algoritmu sledování paprsku.

Abstract

The bachelor's thesis focuses on visual representation of three-dimensional set-up of board game. It describes the possibilities of the visualization of three-dimensional graphics on the Android platform. It deals with principles of advanced rendering methods. The final outcome of the thesis is Android application, which serves as a manager of modules containing implementation of board games. Games are projected using the ray-tracing algorithm.

Klíčová slova

Android, 3d grafika, OpenGL ES, deskové hry

Keywords

Android, 3d graphics, OpenGL ES, board games

Citace

Káčerik Martin: Grafické rozhranie pre doskové hry na platforme Android, bakalářská práce, Brno, FIT VUT v Brně, 2014

Grafické rozhranie pre doskové hry na platforme Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Lukáše Poloka. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Martin Káčerik
21.5.2014

Poděkování

Rád by som poďakoval pánu Ing. Lukášovi Polokovi za vedenie a odborné rady pri riešení tejto bakalárskej práce.

Obsah

1 Úvod

Už samotný názov a zameranie práce núti k zamysleniu: Má to celé vôbec zmysel? Nasadením doskovej hry do elektronického prostredia dochádza k strate aspektu, ktorý osobne považujem za nemenej dôležitú súčasť doskových hier, a to aspektu sociálneho. Medziľudská interakcia vnáša do hry úplne iný rozmer, aj keď to na prvý pohľad nemusí byť zrejmé. Z krátkodobého hľadiska to môže znamenať síce len stratu istého percenta pôžitku (koho by nepotešila výhra „v priamom prenose“ po náročnej partii), z hľadiska dlhodobého to však napomáha modernému trendu presunu medziľudskej sféry z prostredia reálneho do prostredia virtuálneho. Avšak napriek miernej skepse sršiacej z úvodných viet si stačí pripomenúť slová Jana Wericha: „Televize není nepřítel, televize se dá vypnout.“ Keby bol vo svojej dobe tušil, mohol by k televízii doplniť aj všetky iPhone, Nexus či Galaxy, ktoré dnes máme všade so sebou.

Na druhú stranu – máme ich všade so sebou. Len ťažko si môžeme predstaviť, že cestou zo školy či práce domov si vo vozidle mestskej hromadnej dopravy s kamarátom pripravíme a rozohráme majstrovskú partiu šachu alebo nebodaj hry typu A Game of Thrones (s možnosťou dĺžky trvania cez desať hodín). Čo nám však bráni spustiť naše mobilné zariadenie a odohrať za krátku cestu 2 ťahy koňom alebo zvládnuť piškvorky či námornú bitku, aj keď práve nemáme papier a ceruzky? Druhým, najmä pre mladých a študentov nezanedbateľným argumentom je cenová dostupnosť. Cena jednej základnej hracej sady sa šplhá vysoko v desiatkach eur, nehovoriac o dostupných rozšíreniach. Zhrnuté a podčiarknuté – myslím si, že to zmysel má.

Nasledovná kapitola stručne popisuje vývoj doskových hier od minulosti až do dneška a uvádza príklad, ako môže vyzerať grafické rozhranie takejto hry v mobilnom telefóne. Kapitola 3 zhŕňa smery, ktorými sa možno vydať pri hľadaní cesty ku zobrazovaniu trojrozmernej grafiky pomocou mobilných zariadení. Štvrtá kapitola popisuje proces návrhu aplikácie počnúc užívateľským rozhraním a končiac plánovanými použitými technológiami. Piata kapitola sa snaží priblížiť procesy nutné k priebehu aplikácie a cestu k ich finálnej implementácií. Záverečná kapitola obsahuje zhrnutie práce a možnosti nasledovného vývoja.

2 Doskové hry

Doskové hry – pojem zastrešujúci pod sebou nespočítateľné množstvo hodín zábavy, tréningu mysle, strategických či rétorických schopností celého ľudstva skrz tisícročia. Napriek rôznorodosti pravidiel, herných mechanizmov a herných cieľov sú všetky spojené jediným základným kameňom, ktorý im dal ich meno, a to hracou doskou.

Svojím prevedením sa doska líši od hry k hre, vždy sa však jedná o väčšiu či menšiu abstrakciu reálneho sveta reprezentujúcu mesto, budovu, ostrov alebo bojisko. Na hracej doske vykonáva hráč svoje akcie vyjadrené pomocou hracích figúr, ktoré svojou vizážou stvárňujú svoju funkciu, či už ako miniatúry zvierat, predmetov alebo personifikáciou ľudského tela.

2.1 Doskové hry v priebehu vekov

Kým prvé doskové hry z antických čias boli vo veľkej väčšine určené pre dvoch hráčov, stvárňovali súťaž dvoch táborov (či už v konfliktnej – Go, alebo mierovej forme – Senet, viac viď. [1]) a na hru samotnú stačil súbor herných figúr a vhodný hrací plán, postupom času sa komplexnosť hier rozširovala všetkými smermi. Pridaný faktor náhody vo forme hodu kockou, rozšírenie možností na ovplyvňovanie priebehu hry vo forme kariet, utajovanie časti informácií pred druhým hráčom či možnosti obchodovania s hernými surovinami viedli k širokej diverzifikácii na hernom trhu.

V súčasnosti existuje na tisíce rôznych hier, len ich kompletná kategorizácia je možná z mnohých hľadísk a jej rozsah by hlboko presahoval priestor určený pre túto časť kapitoly. Mnoho hier sa vo veľkej miere prelína cez viacero kategórií, preto sa pokúsim iba o hrubé zatriedenie na základe faktov relevantných pre následné riešenie problematiky hernej scény.

2.1.1 Rozdelenie na základe herného princípu

Jednotlivé skupiny hier spája obdobná herná logika, ktorá napriek skutočnosti, že sa aj v rámci skupiny môže znateľne odchyľovať od ideového priemeru, nám napomáha uvedomiť si rôzne fakty a komplikácie spojené s jej neskoršou implementáciou. Pri jednotlivých triedach je vždy uvedená ako príklad hra, ktorá je komunite hráčov a väčšinou aj laickej verejnosti všeobecne známa.

Abstraktné strategické hry

- tento typ hier nie je nijako tematicky lokalizovaný, aspekty náhody a šťastia sú v maximálnej možnej miere minimalizované [2]
- súbor pravidiel je jednoduchý, jasne špecifikovaný a o víťazovi rozhodujú schopnosti a skúsenosti hráča
- šach, Reversi, čínske Go alebo vikingský Tafl

Hry nemeckého typu

- opäť sa jedná o hry s jednoduchými pravidlami, sú však už tematicky viazané, ovplyvňované náhodou, môžu sa využívať herné suroviny aj karty s akciami [3]
- napriek faktu, že interakcia hráčov je na vysokej úrovni, významným spoločným rysom je skutočnosť, že prakticky nedochádza k eliminácii hráčov počas hry

- Osadníci z Katanu, Carcassone, Puerto Rico

Pretekárske hry

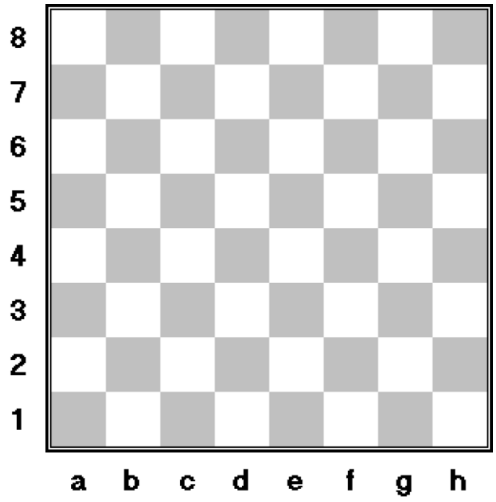
- myšlienka je jednoduchá, cieľom je usporiadať svoje herné figúry do finálnej pozície rýchlejšie ako súper
- náročnosť hry sa mení podľa miery možností strategických zásahov do vývoja od nulových nárokov na schopnosti hráča až po nutnosť plánovania každého ťahu [4]
- Hady a rebríky, Človeče nehnevaj sa, backgammon

2.1.2 Rozdelenie podľa typu hracej dosky

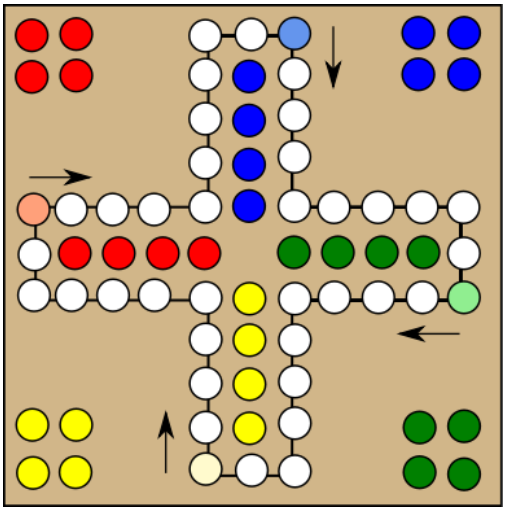
Na základe všeobecných čŕt jednotlivých typov hracích dosiek som vytvoril toto rozdelenie, ktoré reprezentuje následné požiadavky pri ich elektronickej interpretácií

Pravidelné nemenné hracie dosky

Doska je charakteristická tvarom a počtom jednotlivých polí a je pre každú partiu hry identická. Na základe jednoduchých pravidiel je možné ju jednorazovo vygenerovať a donekonečna používať. Typickým zástupcom je šachovnica (Obrázok 2-1) alebo doska pre hru Človeče nehnevaj sa v tvare kríža (Obrázok 2-2).



Obrázok 2-1 Šachovnica[1]



Obrázok 2-2 Človeče nehnevaj sa[2]

Nepravidelné nemenné hracie dosky

Komplexnejšie hracie dosky síce obsahujú, podobne ako už uvedené príklady, jednotlivé herné polia pre umiestnenie figúr, majú ich však rozmiestnené nepravidelne až chaoticky. Vo veľkej miere sa jedná o dopredu daný tematický či príbehový koncept celkovej grafiky hracej plochy. Pri

[1] Obrázok dostupný na http://www.regencychess.co.uk/blog/wp-content/uploads/2012/05/empty-numbered-chess-set.gif
[2] Obrázok dostupný na http://upload.wikimedia.org/wikipedia/commons/thumb/9/91/Menschenaergern.svg/386px-Menschenaergern.svg.png

implementácií takejto dosky je nutné samostatne presne definovať všetky jednotlivé časti. Ako príklad vhodne poslúžia dosky z hry Shadows of Camelot (Obrázok 2-3) alebo A Game of Thrones.

Obrázok 2-3 Shadows over Camelot[3]

Pravidelné meniace sa hracie dosky

Tento typ dosky je reprezentovaný dopredu danými možnými hernými poľami, ktorých tvar je vzájomne kompatibilný a pre každú jednotlivú partiu je z týchto polí náhodne zostavená hracia plocha. Nezriedka sa stáva, že hracia plocha na začiatku prakticky neexistuje a hráči ju sami budujú v priebehu hry. Na tejto báze sú postavené napr. hry Osadníci z Katanu (Obrázok 2-4) alebo Carcassone (Obrázok 2-5).

Obrázok 2-4 Osadníci z Katanu[4]

Obrázok 2-5 Carcassone[5]

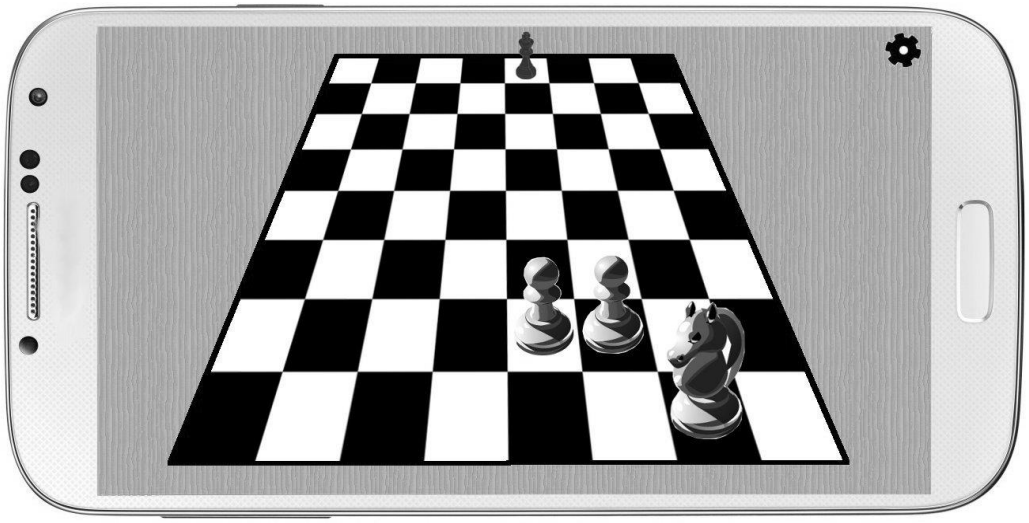
[3] Obrázok dostupný na https://images.funagain.com/illus/huge/15117.jpg
[4] Obrázok dostupný na http://markmeynell.files.wordpress.com/2011/03/game-catan-board.jpg
[5] Obrázok dostupný na http://futureblue.files.wordpress.com/2014/01/carcassonne-board.jpg

2.2 Užívateľské rozhranie

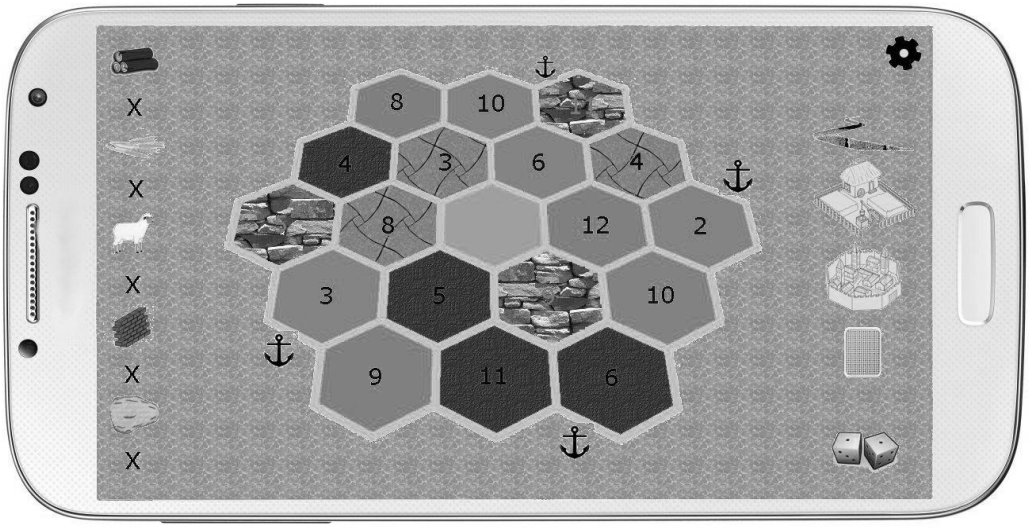
Vhodne navrhnuté užívateľské rozhranie je esenciálnou súčasťou úspechu aplikácie a je nevyhnutné nepodceniť žiadnu jeho časť.

Rozhranie jednoduchých hier s nízkym počtom herných aspektov, ako sú karty či množstvo figúr sa dá navrhnúť pomerne rýchlo a prehľadne aj pre malú plochu inteligentného telefónu či tabletu. S narastajúcim počtom sledovaných veličín ako sú herné suroviny, útočná a obranná sila jednotlivých figúr alebo zoznam vlastnených predmetov a s narastajúcou komplexnosťou herného plánu, ktorého rozmery v stolnej verzii sa bežne pohybujú v desiatkach centimetrov, sa však hra zobrazená na menšej ploche môže stať výrazne neprehľadnou.

Na obrázku (Obrázok 2-6) môžeme vidieť schematicky načrtnuté možné rozhranie partie šachu, ktoré je prehľadné a ľahko pochopiteľné, zatiaľ čo obrázok (Obrázok 2-7) zobrazuje možnosť zobrazenia scény hry Osadníci z Katanu, kde už je obrazovka rozdelená na 3 časti v smere zľava zobrazujúce stav herných surovín, mapu osídľovaného ostrova a možné herné akcie.



Obrázok 2-6 Rozhranie pre šach



Obrázok 2-7 Rozhranie pre Osadníkov z Katanu

3 3D grafika a Android

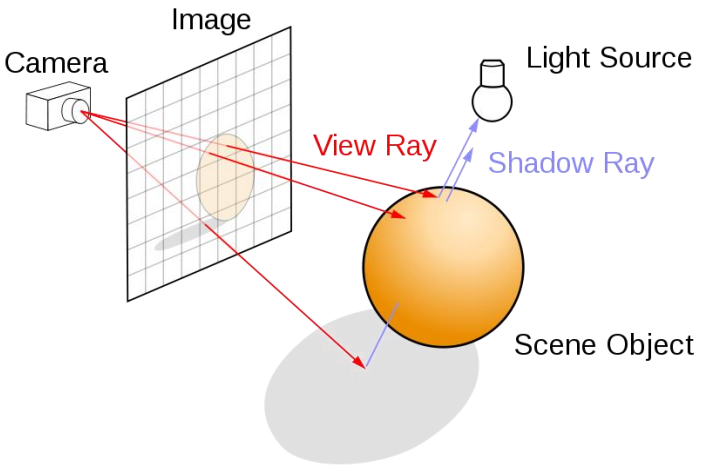
Keďže cieľom tejto práce je vytvoriť plnohodnotnú 3D aplikáciu pre platformu Android, je nutné sa zamyslieť nad vhodnými algoritmami vykresľujúcimi trojrozmerné prostredie a nad možnosťami ich implementácie v prostredí tohto operačného systému.

3.1 Statická obmedzená scéna

Napriek pokroku v hardvérovej oblasti bežne dostupné mobilné zariadenia stále nemôžu byť a ani nie sú vybavené výpočtovými schopnosťami osobného počítača. Preto treba citlivo zvážiť ponúkané možnosti. Pri riešení problematiky zobrazovania scény doskovej stolnej hry však v porovnaní s inými hrami v troch rozmeroch disponujeme pomerne značnou výhodou.

Naše prostredie reprezentuje presne ohraničené prostredie a zároveň je vo vysokej miere statické. To nám poskytuje možnosť využiť zobrazovacie metódy, ktoré poskytujú vysoko kvalitné a realistické zobrazenie scény, ale pri súčasných možnostiach by neboli schopné zaistiť vykreslenie dostatočného množstva obrázkov za sekundu v reálnom čase pre zachovanie plynulosti pohybu v plne dynamickej hre.

Medzi algoritmy umožňujúce realistické zobrazovanie 3d scény patria napríklad algoritmus sledovania lúča (angl. *ray tracing*) a jeho zlepšené verzie v kombinácií s vhodným osvetľovacím modelom alebo algoritmus radiozita (angl. *radiosity*).



Obrázok 3-1 Sledovanie lúča[6]

3.1.1 Sledovanie lúča

V zjednodušenej forme je princíp, akým vnímame svet pomerne prostý. Zo zdroja (napr. slnka) vychádza lúč svetla, ktorý sa odrazí od povrchu predmetu do nášho oka, kde je následne interpretovaný. Algoritmus sledovania lúča je založený na spätnom princípe – z oka (kamery) je skrz

[6] Obrázok dostupný na http://upload.wikimedia.org/wikipedia/commons/thumb/8/83/Ray_trace_diagram.svg/875px-Ray_trace_diagram.svg.png

priemetňu („okno“) do scény vyslaný lúč pre každý pixel tejto priemetne. V okamihu, keď lúč narazí v zobrazovanej scéne na objekt, vyhodnotí sa v danom bode jeho osvetlenie, čo v konečnom dôsledku vyjadruje jeho farbu. Následne je tento bod na základe vypočítaných hodnôt vykreslený [5]. Celý princíp je zobrazený na (Obrázok 3-1).

„View ray“ na obrázku reprezentuje primárny lúč vrhnutý z kamery, *„Shadow ray“* zase sekundárny lúč odrazený do zdroja svetla.

Priesečníky lúča s objektmi v scéne

Táto podkapitola bola prevzatá z [5].

Pokiaľ chceme vyslať do scény lúč, musíme vedieť, po akej trase sa bude pohybovať. Keďže je lúč vysielaný z jedného bodu, môže byť reprezentovaný polpriamkou. Parametrické vyjadrenie polpriamky je vo vektorovom tvare dané rovnicou

$$\vec{\boldsymbol{p}}(t) = \vec{o} + t\vec{d} \tag{3-1}$$

Kde $\vec{o}$ je počiatočný vektor, $\vec{d}$ je smerový vektor priamky (ktorý je možné určiť rozdielom dvoch bodov patriacich priamke), oba dané trojrozmernými súradnicami, t je ľubovoľný parameter pre ktorý chceme určiť súradnice vektoru $\vec{\boldsymbol{p}}$.

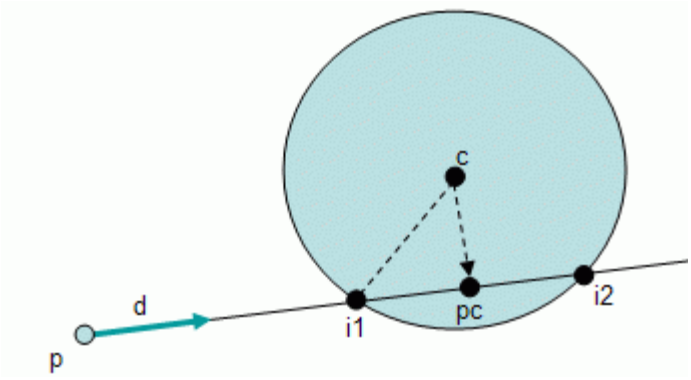
Povrch gule je vo vektorovom tvare vyjadrený rovnicou

$$(\vec{p} - \vec{c})(\vec{p} - \vec{c}) - R^2 = 0$$

kde $\vec{c}$ je stred gule, R je jej polomer a $\vec{\boldsymbol{p}}$ je hľadaný bod. Priesečník lúča s guľou teda získame riešením sústavy dvoch rovníc, čo po dosadení rovnice (3-1) vedie k riešeniu kvadratickej rovnice

$$\left(\vec{d} \cdot \vec{d}\right)t^2 + 2\vec{d}(\vec{o} - \vec{c})t + (\vec{o} - \vec{c})(\vec{o} - \vec{c}) - R^2 = 0$$

ktorej riešením je buď prázdna množina, jeden bod reprezentujúci miesto dotyku lúča na povrchu gule alebo ma riešenia dve, vstupný a výstupný bod z gule (Obrázok 3-2).



Obrázok 3-2 Priesečník lúča s guľou[7]

Priesečník lúča s trojuholníkom je zdanlivo komplikovanejší. Tri nekolineárne body $\vec{a}$, $\vec{b}$, $\vec{c}$ tvoriace vrcholy trojuholníka zároveň definujú rovinu, ktorá sa dá vyjadriť pomocou barycentrických súradníc ako

$$\vec{\boldsymbol{p}}(\alpha, \beta, \gamma) = \alpha\vec{a} + \beta\vec{b} + \gamma\vec{c};\ \alpha + \beta + \gamma = 1 \tag{3-2}$$

kde $\vec{\boldsymbol{p}}$ je bod ležiaci v danej rovine. Bod $\vec{\boldsymbol{p}}$ leží v trojuholníku práve a len vtedy, keď pre jeho barycentrické súradnice platia nerovnosti

$$0 < \alpha < 1; 0 < \beta < 1; 0 < \gamma < 1.$$

[7] Obrázok dostupný na http://www.lighthouse3d.com/wp-content/uploads/2011/03/raysphere3.gif

Pokiaľ sa jedna súradnica rovná nule, nachádza sa bod na hrane, ak sa nule rovnajú súradnice dve, bod leží na vrchole trojuholníka.

Hodnotu barycentrických súradníc môžeme získať výpočtom rovníc

$$\alpha = \frac{O(\Delta\vec{\boldsymbol{p}},\vec{b},\vec{c})}{O(\Delta\vec{a},\vec{b},\vec{c})}; \beta = \frac{O(\Delta\vec{a},\vec{\boldsymbol{p}},\vec{c})}{O(\Delta\vec{a},\vec{b},\vec{c})}; \gamma = \frac{O(\Delta\vec{a},\vec{b},\vec{\boldsymbol{p}})}{O(\Delta\vec{a},\vec{b},\vec{c})}$$

v ktorých je v čitateli vždy obsah tvorený bodom $\vec{\boldsymbol{p}}(\alpha,\beta,\gamma)$ a vždy dvoma rôznymi vrcholmi trojuholníka a v menovateli obsah celého pôvodného trojuholníka.

Po zlúčení rovníc (3-2) získame rovnicu vyjadrujúcu bod v rovine vo forme

$$\vec{\boldsymbol{p}}(\beta,\gamma) = \vec{a} + \beta(\vec{b} - \vec{a}) + \gamma(\vec{c} - \vec{a})$$

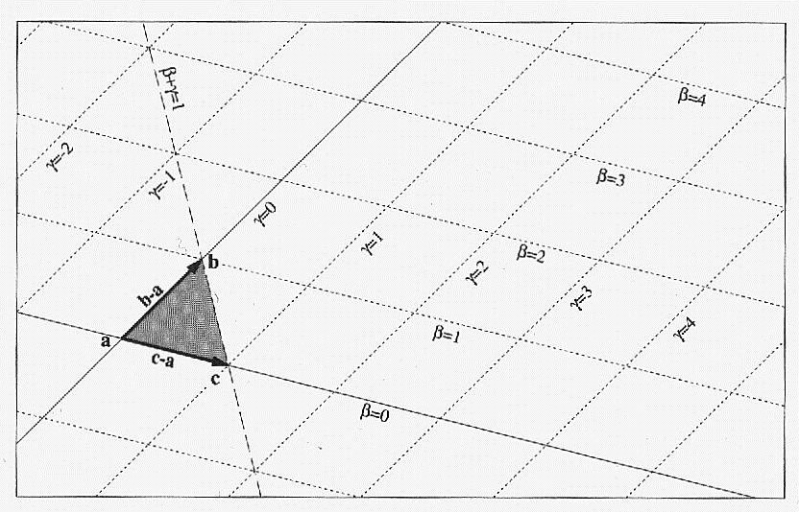
čo vedie k zjednodušeniu sústavy nerovníc podmieňujúcich príslušnosť bodu trojuholníku na

$$\beta + \gamma < 1; 0 < \beta; 0 < \gamma$$

a zároveň po dosadení rovnice (3-1) vedie k rovnici

$$\vec{o} + t\vec{d} = \vec{a} + \beta(\vec{b} - \vec{a}) + \gamma(\vec{c} - \vec{a}), \qquad (3\text{-}3)$$

ktorej riešením je bod prieseku vyslaného lúča a roviny (Obrázok 3-3)[5].



Obrázok 3-3 Trojuholník v rovine vyjadrenej rovnicou (3-3)

Pre vyriešenie rovnice (3-3) potrebujeme však poznať hodnotu súradníc β a γ a parametru t, ktoré získame z prepisu tejto rovnice do tvaru pre 3 rôzne body a dostaneme

$$\begin{bmatrix} a_x - b_x & a_x - c_x & d_x \\ a_y - b_y & a_y - c_y & d_y \\ a_z - b_z & a_z - c_z & d_z \end{bmatrix} \begin{bmatrix} \beta \\ \gamma \\ t \end{bmatrix} = \begin{bmatrix} a_x - o_x \\ a_y - o_y \\ a_z - o_z \end{bmatrix}.$$

Riešením rovnice (napr. pomocou Kramerovho pravidla) získame požadované hodnoty.

Osvetľovací model

Po určení bodu, ktorý sa má vykresliť je ešte nutné určiť jeho farbu. Tá samozrejme závisí od použitého materiálu a textúry telesa, ktoré sú dané, ale aj od svetla dopadajúceho na povrch vykresľovaného telesa. Osvetlenie v trojrozmernej scéne sa simuluje použitím osvetľovacích modelov, ktoré sa snažia čo možno najvernejšie napodobniť správanie sa svetla v reálnom priestore.

Napriek známym nedostatkom je už dlhé desaťročia v praxi rozšírený Phongov osvetľovací model, ktorý síce nie je fyzikálne úplne správny, poskytuje však dostatočne kvalitné výsledky za malú cenu v podobe výpočtových zdrojov.

Osvetlenie podľa Phongovho modelu je dané zlúčením troch svetelných zložiek, a to okolitého svetla, ktoré je simuláciou globálneho osvetlenia scény, difúzneho svetla, ktoré sa od povrchu odráža do všetkých smerov a zrkadlového svetla, ktoré určuje odlesk svetla v danom bode [6].

Podľa miery požadovanej realistickosti scény sa môžu použiť aj komplikovanejšie modely, napríklad upravený Blinnov-Phongov osvetľovací model alebo fyzikálne správny Torranceov-Sparrowov model.

3.1.2 Radiozita

Princíp tejto metódy je diametrálne odlišný od metódy sledovania lúča. Spočíva v predpoklade dodržania zákonu zachovania energie v uzavretom systéme – zobrazovanej scéne – zloženom z plôch, ktoré svetlo pohlcujú, odrážajú alebo emitujú. Tento zákon sa dá vyjadriť rovnicou pre každú plochu v scéne, ktorá každú plochu ohodnotí adekvátne jej vlastnostiam. Toto ohodnotenie je zároveň výstupom radiačnej metódy, takže neponúka samotné zobrazenie – to je nutné vytvoriť pomocou inej metódy, ako možnosť pripadá aj metóda sledovania lúča.

Radiozita je považovaná za prvý algoritmus správne zohľadňujúci nepriame – odrazené svetlo od ostatných objektov [7].

3.2 OpenGL ES

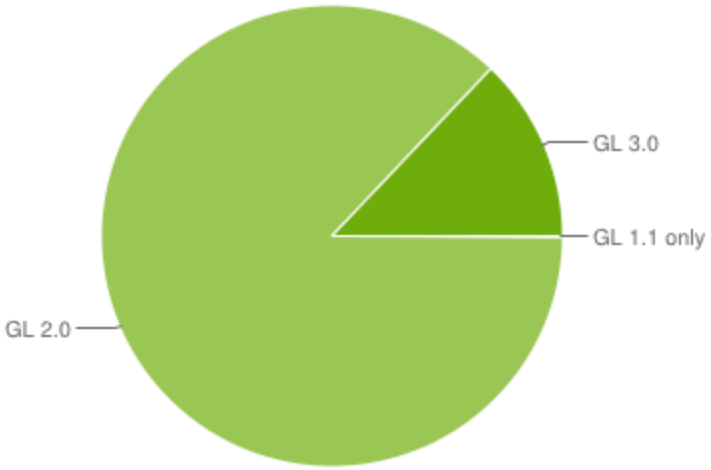
Multiplatformné API OpenGL® určené na podporu zobrazovania dvoj a trojrozmernej grafiky si našlo priestor aj na platforme Android, a to vo svojej mierne okliešteneJ verzii OpenGL ES, kde skratka ES reprezentuje anglické *embedded systems* – vstavané systémy. Android v rôznych verziách svojho API podporuje nasledovné verzie OpenGL ES [8]:

- OpenGL ES 1.0 a 1.1 – dostupné od verzie Androidu 1.0
- OpenGL ES 2.0 – dostupné od verzie Androidu 2.2
- OpenGL ES 3.0 – dostupné od verzie Androidu 4.3

Vzhľadom na zastúpenie podporovaných zariadení na trhu ku dňu 1.5.2014 (Obrázok 3-4) je zrejmé, že pre možnosti kompatibility so širokou škálou používaných mobilných zariadení je z komerčného hľadiska najvhodnejšie voliť použitie verzie OpenGL ES 2.0.

3.2.1 Zobrazovanie pomocou OpenGL ES 2.0

Možnosť vykresľovania a manipulácie objektov v rámci aktivity bežiacej na pod Androidom s pomocou OpenGL ES nám zaručí využitie dvoch základných prvkov, a to triedy `GLSurfaceView` a rozhrania `GLSurfaceView.Renderer`.

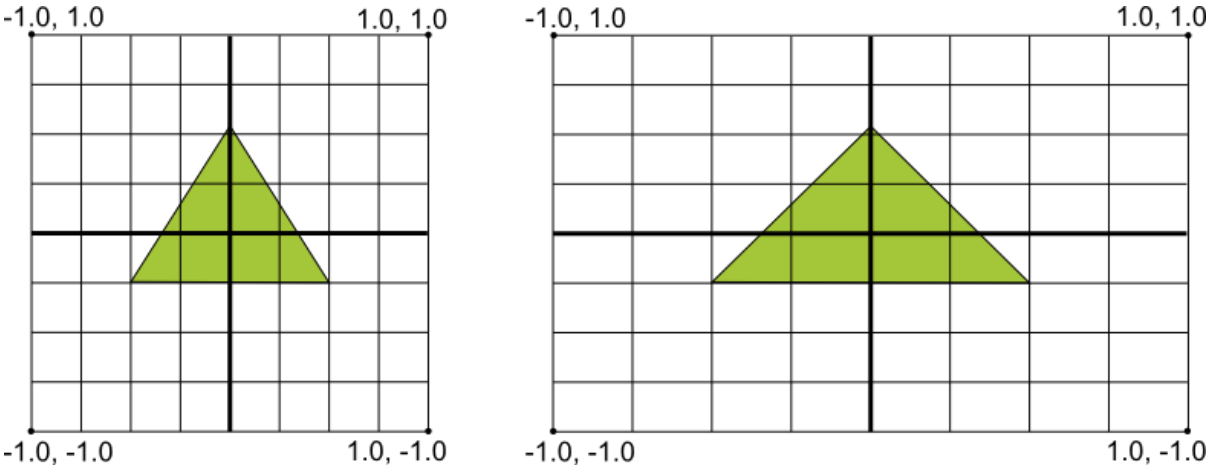


Obrázok 3-4 Podpora zariadení na trhu[8]

`GLSurfaceView` nám poskytuje priestor pre zobrazenie a manipuláciu našich objektov, rozhranie `GLSurfaceView.Renderer` zase zabezpečuje samotné kreslenie objektov do scény. Vyžaduje od nás implementáciu troch metód [8]:

- `onSurfaceCreated()` – volá sa len raz a prebieha tu inicializácia nutných veličín
- `onDrawFrame()` – metóda volaná pri každom prekreslení scény, kreslí scénu
- `onSurfaceChanged()` – metóda volaná pri zmene geometrie scény

Práve na metódu Mapovanie scény na displej zariadenia sa pozrieme bližšie. Za geometriu scény sa v tomto prípade považujú rozmery scény a jej orientácia, preto sa táto metóda vyvolá na začiatku pri tvorbe scény a pri každej zmene parametrov geometrie. Keďže sa platforma Android dodáva na širokej škále zariadení s rôznou veľkosťou obrazovky a od rôznych výrobcov, je nutné zabezpečiť správne zobrazenie scény bez ohľadu na veľkosť a rozlíšenie displeja.



Obrázok 3-5 Mapovanie scény na displej zariadenia[9]

[8] Obrázok dostupný na https://chart.googleapis.com/chart?chs=400x250&cht=p&chd=t%3A0.1%2C87.0%2C12.9&chf=bg%2Cs%2C00000000&chl=GL%201.1%20only|GL%202.0|GL%203.0&chco=c4df9b%2C6fad0c

[9] Obrázok dostupný na https://developer.android.com/images/opengl/coordinates.png

Preto OpenGL ES bez explicitného vyjadrenia uvažuje displej štvorcového tvaru, čo pri zobrazení na bežný displej vedie, ako vidíme na obrázku (Obrázok 3-5), k deformácií tvarov. Preto je nutné zaviesť takzvanú projekčnú maticu, ktorej hodnota sa určí práve pri volaní metódy `onSurfaceChanged()`. Po transformácií zobrazenia pomocou tejto matice je možné zobrazovanie nedeformovaných objektov na obrazovkách rôznych veľkostí [9].

3.2.2 Možnosti implementácie OpenGL ES

Knižnicu OpenGL ES je možné do aplikácie pre Android zahrnúť dvoma spôsobmi, a to buď štandardnou formou skrz framework pre vývoj Android aplikácií alebo pomocou implementácie v natívnom kóde.

Android framework

Táto možnosť nám umožňuje používať funkcionalitu API priamo v rámci projektu v jazyku Java. Požadované referencie do knižníc sú importované a metódy sú implementované bežnou cestou.

Android NDK

NDK – *native development kit* – je súbor nástrojov umožňujúci naprogramovať časť aplikácie v natívnom kóde, napríklad v jazyku C. Panuje všeobecne uznávaný názor, že pri riešení konkrétnej problematiky kódu je nutná jej hlbšia znalosť pre posúdenie, či je vhodnejšie pre jej riešenie zvoliť túto formu implementácie. Z celkového pohľadu sa za vhodných kandidátov považujú výpočtovo náročné operácie, ako sú napríklad fyzikálne simulácie [10].

Pri manipulácií s knižnicou OpenGL ES touto cestou je nutné v domovskom projekte vytvoriť v jazyku Java wrapper (obaľujúci kód) zabezpečujúci vhodnú prácu s prostredím a v natívnom jazyku následne implementovať jednotlivé metódy, ako napríklad už spomenuté metódy rozhrania `GLSurfaceView.Renderer`.

4 Návrh aplikácie

Pri navrhovaní konštrukcie aplikácie som sa rozhodol jej vývoj rozdeliť do troch rozdielnych častí. Prvá časť sa zaoberá manažérom – správcom hier, tvorí hlavné menu, v ktorom sú zobrazené dostupné hry. Druhá časť sa plynulo viaže na prvú, pokrýva správu herných modulov a ich správne načítanie do prostredia spojené s komunikáciou so zobrazovacím jadrom. Poslednú časť tvorí rozhranie samotnej hry spolu so správnou interpretáciou hernej logiky.

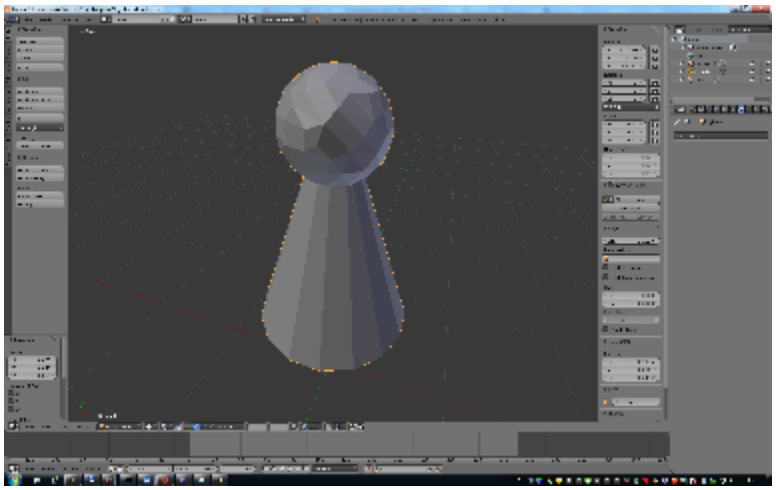
4.1 Rozhranie manažéra

Manažér hier bude tvoriť prvú obrazovku po spustení aplikácie, jeho primárnymi úlohami budú zobrazenie všetkých hier, umožnenie prepínania medzi jednotlivými hrami a umožnenie ich spustenia.

4.1.1 Reprezentácia dostupných hier

Ako abstraktnú reprezentáciu rôznych hier v trojrozmernom prostredí som sa rozhodol použiť objekt, ktorý pozná snáď každý – hraciu figúrku z hry Človeče nehnevaj sa. Už od detstva je v našich hlavách spojená práve s doskovými hrami, čo ju činí vynikajúcim kandidátom na túto pozíciu.

Jej model vytvorený z trojuholníkov som si dopredu pripravil v trojrozmernom modelovacom nástroji Blender, šírenom pod licenciou GNU GPL v2 [11]. Prvá verzia modelu obsahovala bezmála tisíc trojuholníkových plôch, čo je na model, ktorý bude mať po vyobrazení ledva pár centimetrov skutočne príliš mnoho, takže logicky nasledoval proces výraznej redukcie plôch pomocou nástroja *Decimate*.



Obrázok 4-1 Prostredie nástroja Blender s modelom figúrky

Výsledný model som exportoval do štandardného textového súboru typu *OBJ* (viď. 5.1.2).

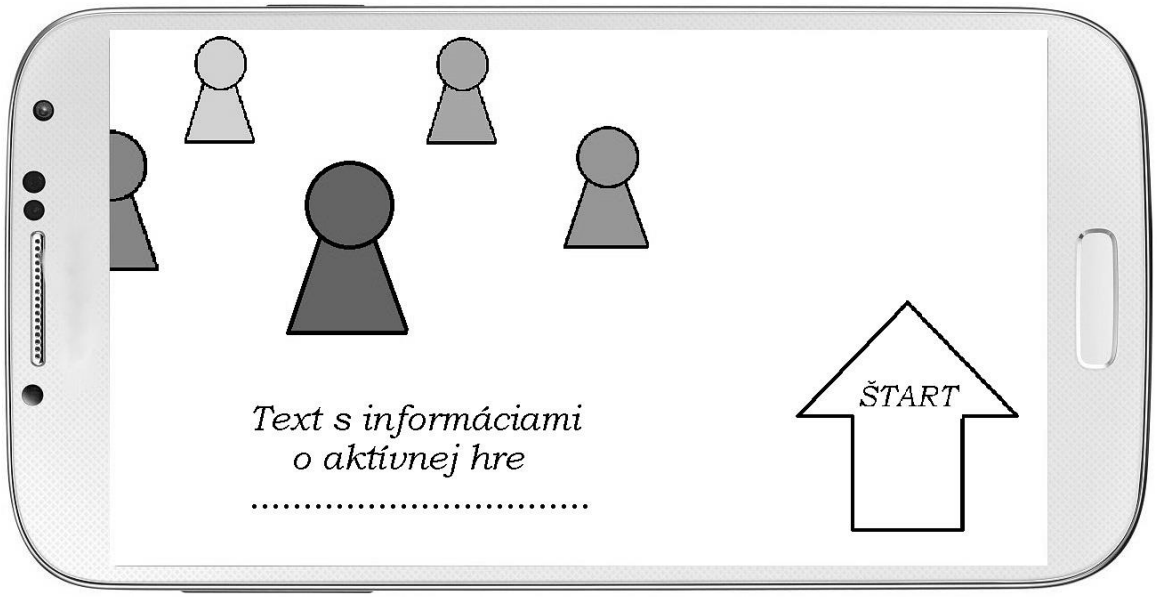
4.1.2 Výber hry z menu

Ako nástroj pre selekciu jednotlivých hier som zvolil trojrozmernú mutáciu kruhového menu, kde na pomyselnej kružnici umiestnenej v rovine súradnicových ôs X a Z budú okolo centra rotovať

jednotlivé figúrky reprezentujúce hry. Rotácia bude prebiehať intuitívne pomocou dotykového displeja. Pri prehliadaní jednotlivých hier sa bude automaticky vykonaný aj jej výber, a to úplne prirodzeným spôsobom – hra, ktorá sa bude nachádzať najviac v popredí bude automaticky zvolená ako výber. Samozrejmosťou bude pre práve zvolenú hru vypísanie základných informácií, podobne ako sme zvyknutí zo škatúľ a obalov doskových hier – odporúčaný počet hráčov, priemerná dĺžka hry či jej náročnosť. Tieto informácie budú k dispozícií v dolnej časti obrazovky.

Ako posledná, ale z praktického hľadiska najdôležitejšia časť je možnosť spustenia hry, ktorá bude prevedená dotykom v pravej dolnej časti displeja.

Všetky nápady sú zhrnuté v obrázku (Obrázok 4-2), ktorý schematicky naznačuje obsah úvodnej obrazovky.



Obrázok 4-2 Schéma užívateľského rozhrania menu

4.1.3 Výsledná grafická podoba

Samotná myšlienka rotácie okolo centra vo mne od začiatku evokovala predstavu vesmíru, preto som sa rozhodol aj celé menu vybaviť vo vesmírnej tematike. Pre nastolenie atmosféry nekonečného vesmíru som vybral metódu zvanú *skybox*, pri ktorej sa celá scéna umiestni do kocky, na ktorej steny sa nanesie vhodná textúra.

Ako vhodnú textúru som si zvolil jednu z mnohých textúr voľne šírených a voľne dostupných na internetovej sieti, ktorá predstavuje pohľad z povrchu neznámej planéty a zachytáva široký vesmír plný galaxií a hmlovín[10].

Hra sa odštartuje ťahaním štartovacieho textu smerom nahor.

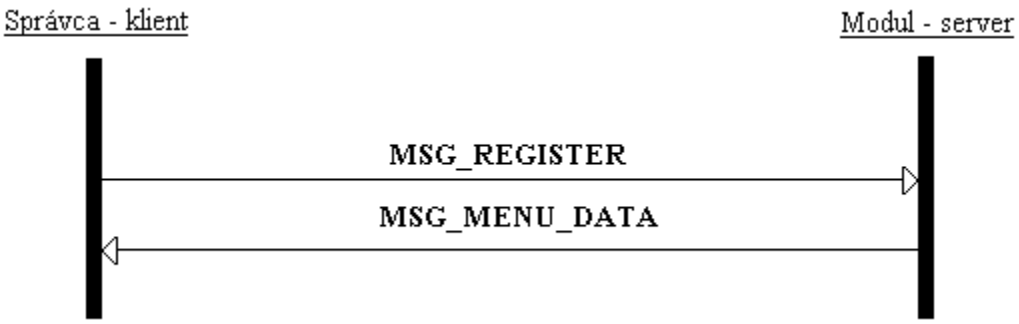
[10] Textúra dostupná na www.custommapmakers.org/skyboxes/zips/mp_moondust.zip

4.2 Systém zásuvných modulov

Táto čisto systémová časť bude mať za úlohu zaobstarať začlenenie jednotlivých modulov, reprezentujúcich samostatné hry, do materskej aplikácie a zabezpečiť spoľahlivú medziprocesovú komunikáciu po spustení hry, a to medzi modulom, v ktorom sa bude nachádzať riadiaca logika hry, a zobrazovacou jednotkou, ktorá bude súčasťou materskej aplikácie.

4.2.1 Načítanie dostupných modulov

Z povahy veci je zrejmé, že k tejto činnosti bude treba pristúpiť medzi prvými, ihneď po štarte aplikácie, keďže na jej výsledku závisí obsah zobrazeného menu navrhnutého v kapitole 4.1.2. Aplikácia bude nútená získať od správcu inštalovaných balíčkov systému Android zoznam dostupných modulov patriacich k hlavnej aplikácií. Táto činnosť bude zabezpečená povinným zaradením modulov pri ich vývoji do dopredu daného menného priestoru. Následne sa podľa vytvoreného protokolu naviaže komunikácia s týmito modulmi a vyžiada sa od nich set informácií požadovaných pre zobrazenie v menu.



Obrázok 4-3 Message flow diagram - načítanie modulu

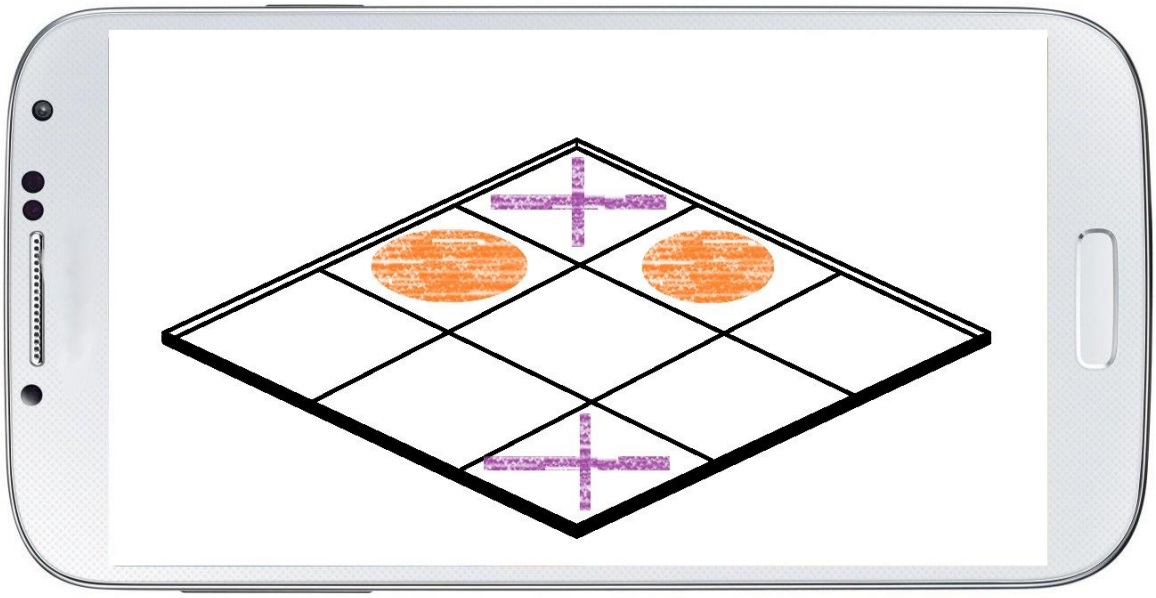
4.2.2 Riadenie procesu hry

Po zvolení a odštartovaní hry nastupuje opäť správca, aby podľa zodpovedajúceho protokolu riadil komunikáciu. Jeho činnosť bude zahájená inicializáciou hernej plochy a odovzdaním informácií o herných figúrach. Následná komunikácia bude závislá od konkrétnej implementovanej hry, k dispozícií bude súbor riadiacich správ, ktorý by mal disponovať dostatočným arzenálom pre generický herný typ s hernou plochou podobnou šachovnici – viď podkapitola 4.3.

Diagram toku správ zodpovedajúci úspešne dohranej implementovanej hre je radený na konci dokumentu ako Príloha B.

4.3 Rozhranie implementovanej hry

Ako hru na demonštráciu celej aplikácie som zvolil pravidlami aj systémom hru veľmi jednoduchú – piškvorky. Jej logika nie je náročná na implementáciu, zároveň však názorne odprezentuje všetky základné úkony, ktoré musí zobrazovanie scény zvládnuť – vykreslenie hernej dosky, identifikácia objektov pomocou dotyku a zobrazovanie herných figúr na zodpovedajúce pozície. Na obrázku (Obrázok 2-1) je schéma rozhrania hry.



Obrázok 4-4 Schéma rozhrania hry piškvorky

5 Implementácia

Vzhľadom na fakt, že v dobe vývoja som mal na testovanie fyzicky k dispozícií zariadenie od firmy Samsung, a to mobilný telefón Samsung Galaxy I9003 s verziou operačného systému Android 2.3.6 (nesúcim označenie *Gingerbread*) som sa rozhodol vyvíjať aplikáciu, ktorá na ňom bude bez problémov fungovať. Preto je zdrojový kód plne kompatibilný s verziou API číslo 10.

Zdrojový kód časti aplikácie v jazyku Java je podľa konvencie triedený do balíčkov s názvom začínajúcim v tomto prípade obrátenou doménou `sk.mkacerik.*;`. Ďalej v texte pri pomenúvaní jednotlivých tried vždy názov celej cesty skrátim o spomínanú časť, jej názov však pre odlíšenie bude vždy začínať bodkou.

5.1 Úvodné menu

Pre relatívne vyššiu mieru interaktivity a vzhľadom na fakt, že zariadenia s Androidom 2.3 už dlhšiu dobu nepatria medzi výkonnostnú špičku mobilných zariadení je celá grafická časť menu spracovaná v jazyku Java s pomocou knižnice OpenGL ES 2.0.

Pri spustení aplikácie sa vyvolá aktivita `.bgActivity.BGActivity`, ktorá po vykonaní formalít (overenie dostupnosti OpenGL ES, nastavení režimu plnej obrazovky) odovzdáva moc do rúk OpenGL vytvorením inštancie triedy `.bgActivity MyGLSurfaceView`. S jej vytvorením ide ruka v ruke aj vytvorenie a následné priradenie k *View* inštancie triedy `.bgActivity MyGLRenderer`, ktorá riadi všetky objekty scény.

Ako bolo proklamované v kapitole 3.2.1, trieda `.bgActivity MyGLRenderer` nevyhnutne implementuje nasledovné 3 metódy.

V metóde `onSurfaceCreated()` sú vytvorené objekty scény a nastavené premenné prostredia OpenGL – napríklad nastavenie hodnoty premennej `GL_CULL_FACE`, čoho následkom sa zakáže vykresľovanie zadných strán trojuholníkov, inými slovami, preskočia sa tie trojuholníky, o ktorých je zrejmé, že ich nebude v danej scéne vidieť.

Metóda `onDrawFrame()` v sebe zahŕňa všetky objekty, ktoré sa budú kresliť. Taktiež je zodpovedná za spočítanie hodnoty matice `mMVPMatrix` (*model view projection matrix*), ktorá je výsledkom vzájomného násobenia projekčnej matice a matice pohľadu, teda umiestnenia kamery v scéne. Pomocou nej sú potom všetky vykresľované objekty scény transformované do správneho pohľadu.

Nakoniec ostáva metóda `onSurfaceChanged()`, ktorá po zavolaní spočíta hodnotu dvoch matíc – projekčnej (viď. 3.2.1) a ortonormálnej pre zobrazenie textu.

5.1.1 Zobrazenie kruhového menu

Pri volaní metódy `onSurfaceCreated()` sa vytvorí inštancia triedy `.menu.MyMenu`, pri jej konštrukcií už musí byť známy počet hier, pre ktoré sa bude menu vytvárať. Riadiacim centrom menu je kružnica z triedy `.menu.Circle`. Táto trieda mimo iné obsahuje 2 esenciálne metódy, konkrétne `calculate()` a `autoRotation()`. Pred volaním metódy `calculate()` sa na základe počtu prvkov kružnica rozdelí na určený počet častí a v tejto metóde sú následne pomocou rovníc

$$x = s_x + R \cdot \cos\omega$$
$$y = s_y + R \cdot \sin\omega$$

spočítané súradnice x, y bodu, na ktorý patrí jeden z prvkov menu. V rovniciach ďalej vystupujú súradnice stredu kružnice s_x ,s_y a R ako polomer kružnice.

Metóda `autoRotation()` zabezpečuje dotočenie menu do základnej pozície po interakcií užívateľa postupným menením uhlu rotácie, čím je zároveň prirodzene znížená rýchlosť rotácie, keď sa prvok blíži k svojej pozícií.

5.1.2 Figúrka – hra v menu

Po vytvorení riadiacej kružnice a spočítaní počiatočných bodov, na ktorých sa zobrazia jednotlivé prvky menu nasleduje načítanie modelu figúrky. Jedna figúrka je v rámci kódu uchovávaná ako objekt triedy `.shape.Figure`. V tejto triede je obsiahnutý jeden statický model tvorený poľom trojuholníkov, vyjadrujúci rozmiestnenie prvkov figúrky všeobecne v priestore a unikátne pole trojuholníkov uchovávajúce konkrétne umiestnenie danej figúrky v priestore.

Keďže vymodelovaný tvar figúrky je exportovaný do súboru typu *OBJ* (formát reprezentujúci 3d model v textovej forme), na základe špecifikácie tohto súborového typu som vytvoril jednoduchý parser, ktorý z tohto textového formátu načíta dáta nutné pre vnútornú reprezentáciu modelu v rámci programu.

Parser je umiestnený do triedy `.objParser.ObjParser` a jeho činnosť je v skratke pozostáva z nasledovných činností. Keďže na začiatku nevieme, z koľkých prvkov sa bude načítavaný model skladať, pre reprezentáciu načítaných štruktúr som zvolil dynamickú štruktúru typu zoznam. Po stanovení štruktúr dochádza k otvoreniu súboru s dátami. Tento je umiestnený v zložke `/assets`, jednej z dvoch štandardných zdrojových zložiek Android aplikácií. Na prístup k nej však potrebujeme ukazovateľ na túto zložku. V prostredí frameworku pre vývoj Android aplikácií na to slúži trieda s názvom `AssetManager`, o vytvorenie jej inštancie sa postará funkcia `getAssets()`. Táto sa však dá volať len v rámci kontextu aplikácie, ktorý je za bežných okolností dostupný len v rámci triedy aktivity. Preto moja trieda `.bgActivity.BGActivity` obsahuje statickú premennú `am`, ktorá umožňuje prístup do zložky `/assets`.

Po otvorení vstupného prúdu pre dáta zo súboru nasleduje ich načítavanie po riadkoch. Dáta v *OBJ* súbore sú reprezentované týmto spôsobom: každý riadok reprezentuje jeden typ údaju. Na jeho začiatku sa vždy nachádza riadiaci symbol nasledovaný dátami. Medzi riadiace symboly, ktoré ma pre moju aplikáciu zaujímajú (kompletný výpis je súčasťou špecifikácie), patria dva, a to „`v`“ a „`n`“ [12].

Symbol „`v`“ (*vertex*) reprezentuje jeden vrchol v modeli, nasledovaný je vždy troma z princípu desatinnými číslami reprezentujúcimi súradnice bodu v trojrozmernej ortonormálnej súradnicovej sústave. Vrcholy sa v mojom projekte ukladajú ako inštancie triedy `.shape.Vertex`, takže pre každý načítaný vrchol je vytvorený objekt tohto typu a je zaradený do zoznamu.

Symbol „`f`“ (*face*) zase zodpovedá jednotlivým plochám modelu. V mojom prípade je nevyhnutné (a pri exporte modelov do formátu *OBJ* pre tento projekt explicitne vyžiadané), aby tieto plochy boli reprezentované trojuholníkom. Pre každý trojuholník sú dáta reprezentované opäť trojicou údajov oddelených medzerou, tento krát to však nie je jednoduchý údaj, je tvorený ďalšími až troma číslami oddelenými lomkou. Na reprezentáciu objektu mi postačuje vždy len prvé číslo z tejto trojice, ktoré vyjadruje index do poľa už načítaných vrcholov. Po parsovaní sú teda k dispozícií tri

indexy pre tri vrcholy trojuholníka. Je nutné si uvedomiť, že vrcholy v súbore typu *OBJ* sú indexované počnúc číslom 1, čo sa líši od štandardného indexovania v rámci jazyka Java (počnúc nulou), takže pri prístupe do zoznamu vrcholov je dôležité zahrnúť posunutie o jednu pozíciu. Trojuholníky sú v projekte vedené v rámci triedy `.shape.Triangle`.

Po načítaní celého modelu sa súbor korektne uzavrie a výsledná štruktúra trojuholníkov je uložená v objete triedy `.shape.Figure` ako reprezentácia jednej figúrky. Keďže v rámci tejto triedy sú uchovávané aj špecifické údaje o hre, dostupné s jednotlivými nahranými hrami, Pre každú hru je vytvorený jeden objekt so samostatným súkromným modelom.

5.1.3 Vykreslenie figúriek

V tomto okamihu máme k dispozícií všetky dáta potrebné k vykresleniu jednotlivých prvkov menu. Keďže sa nejedná o nič iné než množstvo trojuholníkov, o samotné vykreslenie sa stará metóda `drawOneColor()` z triedy `.shape.Triangle` volaná v každom cykle metódy `onDrawFrame()` pomocou obaľujúcej metódy `drawFigure()` z triedy `.shape.Figure` zahŕňajúcej v sebe cyklus pre vykreslenie figúrky ako pomyselného celku.

Zobrazovanie objektov sa v rámci OpenGL uskutočňuje pomocou OpenGL objektu typu `Program`, ktorý v sebe zahŕňa skompilované *shader* (špeciálny program popisujúci, ako je objekt osvetlený a vykreslený) programy a riadi odosielanie dát do grafického vykresľovacieho reťazca, ktorého výsledkom je na displeji zobraziteľná dvojrozmerná scéna ako (na grafickom procesore – GPU) rasterizované zobrazenie trojrozmernej scény. Pre vykreslenie figúrok sú použité dva jednoduché *shadery*.

Vertex shader slúži na renderovanie vrcholov objektu (v tomto prípade trojuholníka). Zdojový kód jeho programu v jazyku OpenGL ES Shading language (ďalej už len ESSL) je uložený v triede `.shader.vertexShader` s názvom `vertexShaderCode`. Jeho obsah je pomerne jednoduchý a krátky, dochádza tu k prepočtu pozícií jednotlivých bodov v scéne, vrátane pozícií dôležitých pre osvetlenie scény, ako sú poloha zdroja svetla alebo poloha oka pozorovateľa. Tieto hodnoty sú pomocou premenných s kvalifikátorom `varying` prenesené do *Fragment shaderu*, kde následne slúžia k výpočtu osvetlenia.

Fragment shader má za úlohu správne vykresliť telo tvaru s adekvátnou farbou/textúrou. Pre zobrazenie figúrok v menu je použitý program `fragmentPhongShading` uložený v triede `.shader.fragmentShader`. Ako už napovedá jeho názov, je v ňom implementovaná mierna variácia Phongovho osvetľovacieho modelu vyhovujúca pre mnou požadované osvetlenie scény. Pomocou štandardných implementovaných funkcií jazyka ESSL (ako `dot()` pre skalárny súčin vektorov či `max()` pre zvolenie vyššej z dvoch daných hodnôt [13]) je prevedený výpočet jednotlivých svetelných zložiek a následne ich správne zobrazenie.

V rámci metódy `onDrawFrame()` sú do *programu* pomocou získaných *handlov* (angl. *handle* – v programátorskom kontexte forma odkazu na prvok/objekt) nahrané jednotlivé údaje vyžadované v *shaderoch*, ako poloha, farba či normála v daných bodoch. Taktiež sú do *programu* vložené údaje z *model view projection* matice, ktorá modifikuje náhľad scény pre oko pozorovateľa. Po odovzdaní všetkých požadovaných dát sa jednotlivé trojuholníky vykreslia štandardnou funkciou `GLES20.glDrawArrays()`.

5.1.4 Rotácia menu a selekcia prvkov

Prehliadanie prvkov v menu je umožnené jeho rotáciou okolo osi rovnobežnej so súradnicovou osou Y a zároveň prechádzajúcou stredom riadiacej kružnice menu.

Pri práci s menu sa dá naraziť na dva rôzne typy rotácie, obe vykonávané na rovnakom princípe, ale iniciované iným zdrojom. Rotácia je založená na menení veľkosti uhla slúžiaceho pre výpočet pozícií pre prvky menu, ktorých súradnice sú získané pomocou metódy `calculate()` zmieňovanej v podkapitole 5.1.1.

Úzko s rotáciou súvisí aj selekcia prvkov z menu, ktorá je vykonaná automaticky natočením prvku menu do takzvanej aktívnej pozície, ktorá sa nachádza v popredí. Určenie aktuálne aktívnej figúry sa automaticky vykoná pri inicializácií menu a aktivita je pridelená figúre s indexom 0, ktorá je na počiatku umiestnená v aktívnej pozícií. Následne pri každom pohybe menu je pomocou metódy `getClosestToActPos()` z triedy `.menu.Circle` na základe vzdialenosti figúr od aktívnej pozície určená nová aktívna figúra.

V každom jednotlivom volaní metódy `onDrawFrame()` je volaná rotáciu obsluhujúca metóda `rotateMenu()` triedy `.menu.MyMenu`. Jej volanie vždy predchádza volaniu zobrazovacej metódy `drawFigure()` a pokrýva oba typy rotácie, ku ktorým v menu dochádza.

Automatická rotácia

Automatická rotácia je vyvolaná faktom, že figúrky nie sú umiestnené na svojich statických piedestáloch určených prvotným výpočtom pozícií. Pre jej určenie sa používa poloha bodu nazvaná v programe ako `activePosition`, v predchádzajúcom aj nasledujúcom texte označovaná pojmom „aktívna pozícia“. Figúrka v menu, ktorá je aktuálne najbližšie aktívnej pozícií je označovaná pojmom „aktívna figúrka“.

V rámci každého volania metódy `rotateMenu()` je vyvolaná aj metóda `autoRotation()` zbežne zmienená v podkapitole 5.1.1. Jej úlohou je pomocou dostupných metód `distance()` a `positionRelativeToX()` z triedy `.shape.Vertex` určiť aktuálne postavenie aktívnej figúrky v priestore. V prípade zistenia faktu, že figúrka sa nenachádza priamo v aktívnej pozícií sa dynamicky vyvoláva zmena rotačného uhla určujúca pohyb pripomínajúci zaparkovanie aktívnej figúrky na aktívnu pozíciu.

Keďže krokovanie pohybu je určené uhlom a prejdená vzdialenosť je teda závislá od polomeru riadiacej kružnice, porovnávanie absolútnej zhody pozície aktívnej figúry s aktívnou pozíciou sa javí ako holý nerozum. Preto sa prvok považuje za zaparkovaný v aktívnej pozícií za predpokladu, že je k nej dostatočne blízko. V rámci tohto projektu som zvolil ako vhodnú maximálnu hodnotu vzdialenosti od aktívnej pozície hodnotu 0,05 jednotky.

Manuálna rotácia

Užívateľskou interakciou dochádza k vyžiadanej manipulácií s menu, ktorá je nadradená automatickému „parkovaniu“ prvkov na ich pozície. Dotykom a vykonaním pohybu v smere požadovanej rotácie v línii s osou X dochádza k úprave rotačného uhla a tým pádom k posunutiu všetkých prvkov v menu.

Zachytenie pohybu na dotykovom displeji je obstarané štandardnou cestou, a to implementáciou metódy `onTouchEvent()` v triede `.bgActivity.MyGLSurfaceView`. Pri zaznamenaní vykonania akcie `ACTION_MOVE` je vyvolaná metóda `setRotationAngle()`, ktorá na základe dĺžky vykonaného pohybu a pomocou empiricky získanej konštanty ovplyvňujúcej

rýchlosť rotácie určí a upraví rotačný uhol. Figúry menu sú s okamžitou platnosťou prenesené do požadovaných pozícií a následne automaticky umiestňované do štandardnej pozície.

5.1.5 Zobrazenie textu v prostredí v GLSurfaceView

Zobrazovanie textu v rámci OpenGL ES pohľadu je náročnejšie, než sa môže na prvý pohľad javiť. Keďže Android SDK ani OpenGL ES neponúkajú žiadnu priamu cestu, zostávajú v zásade štyri možnosti, ako daný text zobraziť:

- Na `GLSurfaceView` zobraziť TextView a vypísať požadovaný text v ňom. Vo väčšine prípadov pomalé a nevhodné, ale v niektorých situáciách použiteľné riešenie.
- Vyrenderovať si požadovaný text do textúry, tú načítať a zobraziť. Riešenie veľmi jednoduché a rýchle, avšak pramálo praktické a prenositeľné.
- Vytvoriť si vlastný kód mapujúci znaky do tzv. „sprajtov“ (angl. *sprites*) – malých dvojrozmerných textúr, z ktorých sa bude následne skladať a zobrazovať text bežnou formou zobrazovania objektov v OpenGL.
- Použiť jednu z dostupných knižníc, správne ju integrovať do projektu a nestarať sa o princípy jej fungovania.

Po dlhšom skúmaní problematiky som vybral niečo medzi treťou a štvrtou možnosťou. Voľne šíriteľný projekt s názvom *Texample2* je implementáciou tretej možnosti, umožňuje načítať a použiť fonty v klasickom formáte *TTF* a taktiež umožňuje ich vypisovanie pomocou vlastných *shaderov* [14].

Po integrácií projektu s jadrom umiestneným do balíčku `.textDraw.*` a po zvládnutí princípu vypisovania textu, ktorý je z používateľského hľadiska jednoduchý – vytvorí sa objekt triedy `GLText`, nastaví sa používaný font a následne v metóde `onDrawFrame()` sa pomocou párových funkcií `GLText.begin()` a `GLText.end()` vytvorí úsek kódu, v ktorom sa pomocou volania viacnásobne preťaženej metódy `GLText.draw()` vykresľuje požadovaný text – som myslel, že práca na tejto časti končí. Bohužiaľ sa však ukázalo, že vybraný projekt má svoje nedostatky ohľadom efektivity vykresľovania a v neskoršej fáze projektu, keď sa to začalo závažnejšie prejavovať, som bol nútený preskúmať zdrojový kód hlbšie a vniesť niekoľko základných optimalizácií, ktoré citeľne zlepšili jej použiteľnosť.

Optimalizácia spočívala v úprave metódy `GLText.draw()` spôsobom, ktorý mal za následok drastické zníženie objemu výpočtu transformačných maticových operácií obmedzením ich volania v nepotrebných prípadoch.

Pre svoju aplikáciu som zvolil font ladiaci s prostredím s menom Coalition[11], voľne použiteľným na nekomerčné účely. Podobne ako model figúrky je uložený v zdrojovom adresári `/assets`.

5.1.6 Skybox

Poslednou implementovanou súčasťou rozhrania správcu je *skybox*, riešenie pomocou textúry simulujúcej nekonečnú rozľahlosť priestoru.

Zdrojové súbory textúry sú uložené vo formáte *PNG* v zložke `/res/raw`, ktorá sa sprístupňuje obdobne ako zložka `/assets` – získaním objektu triedy `Resources`. Trieda

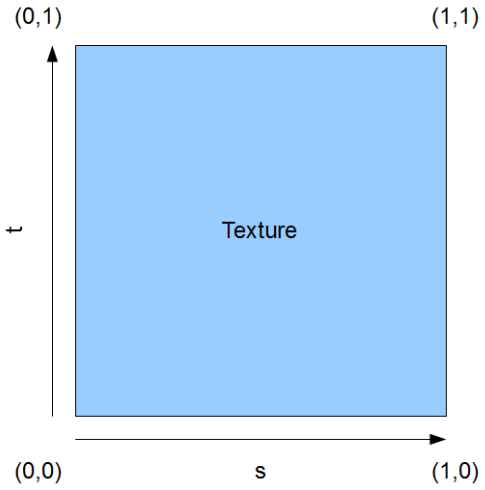
[11] Font dostupný na http://www.dafont.com/font-comment.php?file=coalition

`.bgActivity.BGActivity` obsahuje statickú premennú `rm`, ktorá reprezentuje potrebný objekt volaním metódy `getResources()` v kontexte aktivity. Súbory sú následne dostupné ako prvky triedy `R`.

Po načítaní textúr zo zdrojových súborov do OpenGL pomocou metódy `loadTexture()` z triedy `.shape.Square` je vytvorená nekompletná kocka vhodných rozmerov (pre použitie v menu je to kocka o dĺžke hrán 14 jednotiek), logicky zložená zo štvorcov, na ktoré sa následne budú mapovať textúry. Kocka nie je kompletná z dôvodu šetrenia zdrojmi – v tejto verzii menu sa totiž nachádza len statická kamera, takže zadná strana *skyboxu* nie je nutná.

Proces vykreslenia *skyboxu* je nápadne podobný procesu zobrazovania figúrok menu, keďže sa jedná o rovnaký princíp. Opäť je vytvorený program obsahujúci skompilovaný kód *shaderov* – zatiaľ čo *vertex shader* je podobný ako pri figúrkach, *fragment shader* už nepočíta žiaden osvetľovací model, ale zobrazuje na plôške požadovanú textúru pomocou funkcie jazyka ESSL `texture2d()` – pomocou získaných *handlov* sú do neho odoslané požadované údaje a následne je výsledný raster zaradený do zobrazovanej scény.

Pri mapovaní textúry na dvojrozmernú plochu v OpenGL ES je nutné nezabudnúť zahrnúť jednu nenápadnú, ale zradnú záležitosť. Súradnicová sústava textúry načítanej do OpenGL má počiatok v ľavom dolnom rohu (Obrázok 5-1), zatiaľ čo súradnicová sústava bežnej bitovej mapy má počiatok v ľavom hornom rohu. Tento fakt má za následok, že textúra je načítaná hore nohami a pri jej mapovaní je nutné pretočiť súradnicu *y*. Táto činnosť je prevedená pomocou matíc uložených v premennej s menom `cubeTextureCoordinateData`, ktorá vo svoj prospech využíva fakt, že plocha typu štvorec je v skutočnosti vykresľovaná ako dva trojuholníky v poradí definovanom v poli `drawOrder[]` [15]. Na dané poradie bodov je následne zodpovedne mapovaná textúra podľa definovaných dát matíc s pretočenou súradnicovou osou Y.



Obrázok 5-1 Súradnicová sústava textúry v OpenGL[12]

Pravdou ostáva, že pri použití statickej kamery sa stráca nevyhnutnosť použitia *skyboxu* a postačovala by len jedna textúra zobrazujúca pozadie scény, ale jeho implementáciou zostáva smerom do budúcnosti priestor smerom do budúcnosti pre zvýšenie komplexnosti menu zavedením dynamickej kamery.

[12] Obrázok dostupný na http://www.learnopengles.com/wordpress/wp-content/uploads/2011/09/texture-coordinates-300x300.png

5.2 Zásuvné moduly

Aby mohol zásuvný modul naplniť podstatu svojej existencie, musí byť samostatne šíriteľný, preto sú tieto moduly tvorené samostatnou Android aplikáciou. Keďže v našom prípade poskytne pre modul grafické rozhranie materská aplikácia, nie je nutné ho implementovať ako aktivitu. Pre aplikácie tohto charakteru slúži komponent s názvom služba (angl. *service*). Existujú 2 druhy:

- Štartovaná služba (angl. *started service*) – jej spustenie musí byť explicitne iniciované volaním metódy `startService()`, po jej volaní môže teoreticky bežať donekonečna.
- Viazaná služba (angl. *bound service*) – aplikácia vyžadujúca jej službu je nútená naviazať sa pomocou `bindService()`, čo jej poskytuje klient-server rozhranie pre komunikáciu, takáto služba beží, len pokiaľ má na seba naviazaného klienta [16].
  Požiadavkám pre zásuvný modul doskovej hry vyhovuje viazaná služba.

5.2.1 Nadviazanie komunikácie

Vytvorenie viazanej služby priamo pomocou AIDL (Android Interface Definition Language) je vzhľadom na fakt, že s modulom bude komunikovať ako klient vždy len náš manažér, zbytočne komplikované. Preto použijeme na jeho vytvorenie triedu `Messenger`.

Krátko po spustení si manažér pomocou triedy `PackageManager` zo zariadenia získa zoznam dostupných služieb a z neho pomocou porovnávania názvov (určených v súbore `AndroidManifest.xml`) vyfiltruje herné moduly. Aby bol modul akceptovaný manažérom, musí byť vedený v mennom priestore `sk.mkacerik.bgService.*`.

Na základe tohto názvu a za prispenia objektu triedy `Intent` sa volaním metódy `bindService()` materská aplikácia naviaže na službu. Pri úspechu je zahájená komunikácia odoslaním správy `MSG_REGISTER` službe, na ktorú ona vzápätí reaguje odoslaním správy `MSG_MENU_DATA` obsahujúcou dáta nutné k zobrazeniu informácií o hre v klientovi.

Systém reakcií na prijaté správy je zaistený implementáciou metódy `handleMessage()` z triedy `Handler`, ktorá musí obsahovať reakcie na všetky dostupné správy. Dáta sú k správam pripájané ako inštancie triedy `Bundler`, ktoré je správa schopná prenášať.

Po zahájení hry v manažéri je vytvorením inštancie triedy `.menu.GameLoader` vytvorená nová aktivita zobrazujúca scénu hernej plochy a do modulu je odoslaná správa s názvom `MSG_GET_BOARD` požadujúca informácie o rozmeroch hernej plochy a po jej obslúžení správa `MSG_GET_TOKENS`, odovzdávajúca informácie o herných figúrach. Týmto je hra odštartovaná.

Následne prebieha vzájomná výmena správ o umiestňovaní herných figúrok po ploche a prebieha vyhodnocovanie hernej situácie. Hra je ukončená po tom, čo server na správu o umiestnení reaguje správou `MSG_HE_WON`.

5.2.2 Premenné a funkcie modulu

Pre správne fungovanie komunikácie musí modul nevyhnutne implementovať štandardné správy s odpoveďami na uvedené prichádzajúce požiadavky. Ku každej správe sú priradené údaje alebo funkcia (ktorá tieto údaje poskytne), ktoré naplnia dátovú časť správy.

- `MSG_REGISTER -> MSG_MENU_DATA`
  - `String name;`

    - `String playerCount;`
    - `String avgPlaytime;`
    - `String difficulty;`
    - Správa odovzdá uvedené údaje pre herné menu
- `MSG_GET_BOARD -> MSG_BOARD`
    - `integer size;`
    - Táto počiatočná verzia herného rozhrania predpokladá štvorcovú hernú plochu typu šachovnica, kde `size` určuje počet herných políčok na jednej strane.
- `MSG_GET_TOKENS -> MSG_TOKENS`
    - `integer count;`
    - `float[] model;`
    - `count` určuje počet rôznych figúr, ktorými modul disponuje, `model` obsahuje trojuholníkovú reprezentáciu hernej figúry
- `MSG_TOKEN_X_POS, MSG_TOKEN_Y_POS -> MSG_X_ACK, MSG_Y_ACK`
    - `foo acceptAction();`
    - funkcia nastaví do odpovede typ akcie, ktorý sa očakáva na základe výberu
- `MSG_X_ACTION, MSG_Y_ACTION -> MSG_X_OK, MSG_Y_OK`
    - `boolean wonCondition();`
    - správa nesie informáciu o tom, či vykonaná akcia spôsobila ukončenie hry v prospech jedného z hráčov
- `MSG_WHO_WON -> MSG_X, MSG_Y`
    - `String player;`
    - nesie informáciu o tom, kto hru vyhral, pre prípad, že hráč môže vyhrať hru mimo svoj ťah

Vhodná implementácia uvedených správ postačuje na riadenie jednoduchej hry na hracej doske typu šachovnice pre dvoch hráčov.

5.3 Scéna hry

Samotná herná scéna je zobrazovaná pomocou algoritmu sledovania lúčov, popísaného v kapitole 3.1.1. Je implementovaná s pomocou NDK (viď. 3.2.2) v jazyku C++. Časti kódu sú prevzaté z [5], ktorá slúžila ako hlavný zdroj vedomostí a informácií o technológii sledovania lúča. Scéna je zobrazovaná v rámci novej aktivity ako sledovaním lúča získaná bitová mapa. Keďže vieme dopredu povedať, že scéna sama o sebe je čisto statická a bez zásahu užívateľa v nej nedochádza k zmenám, nie je nutné získavať informácie o jej vzhľade neustále, napríklad formou cyklu bez prestávky vrhajúceho lúče do scény. Plne dostačujúce je vykonať obnovenie stavu scénu vždy vtedy, keď nám prichádzajúca správa odovzdá informáciu o zmene.

5.3.1 Základné prvky ray-traceru v kóde

Najprimitívnejšia štruktúra – vektor – je v projekte implementovaná ako samostatná trieda nesúca názov `Vector3`, ktorej zdrojový kód (ako aj väčšiny ostatných základných častí) bol prevzatý z [5]. Keďže objekt vektoru je používaný v absolútnej väčšine výpočtov, je nutné zaistiť efektivitu aritmetických operácií nad ním vykonávaných. Preto sa v čo najväčšej miere využívajú konštanty a vkladané (angl. *inline*) funkcie.

Svojou vlastnou triedou s názvom `rgb` disponuje aj farba reprezentujúca farebný model RGB.

Všetky objekty a plochy zobrazované v scéne sú potomkami triedy `Shape`, ktorá vyžaduje implementáciu dvoch virtuálnych metód neskôr používaných pre výpočet priesečníku lúča s týmto objektom.

Pre výpočet osvetlenia v scéne som opäť zvolil osvedčený Phongov osvetľovací model.

5.3.2 Zobrazovanie dosky a selekcia objektov

Zobrazenie hracej dosky je závislé od získaného parametru určujúceho jej rozmery. Po získaní tohto čísla je vypočítaný rozmer jedného poľa hracej dosky (aby sa ako celok zmestili do záberu) a doska je vytvorená ako dvojrozmerné pole objektov triedy `BoardUnit`. Každý objekt hracej plochy obsahuje referenčný bod, pomocou ktorého môže naň byť umiestnená hracia figúrka.

Po zobrazení scény očakáva *ray-tracer* modely všetkých hracích figúrok, ktoré eventuálne môžu zasiahnuť do hry. Tieto modely sú uložené do inštancií triedy `Mesh`. Ich rozmery a poloha voči referenčnému bodu sa musia prispôsobiť veľkosti hracej scény tak, aby bolo možné ich korektné zobrazovanie na hraciu plochu.

Po prijatí niektorých typov správ je nevyhnutné vybrať vhodný prvok na scéne, s ktorým má byť prevedený nejaký typ akcie. Existuje viacero možností, ako sa vyrovnať s nutnosťou identifikácie objektu v trojrozmernej scéne. Keďže sa však nachádzame v prostredí *ray-tracera* a máme k dispozícii všetky jeho metódy, bolo by absurdné zvažovať inú metódu ako je vrhnutie lúča.

Výpočet priesečníkov lúčov s objektmi v scéne je pre každý navrhnutý tvar či model implementovaný v dvoch rôznych polymorfných metódach nesúcich názvy `hit()` a `shadowHit()`. Rozdiel je, najmä z hľadiska výpočtovej náročnosti markantný. Zatiaľ čo metóda `hit()` určuje pre každý priesečník aj jeho farbu, `shadowHit()` len vráti hodnotu typu `boolean` s informáciou o zásahu objektu.

Nezávisle od konkrétneho požiadavku je preto postup zisťovania vybratého vždy identický – do získaného bodu sa z pohľadu kamery vrhne jeden lúč. Metóda `shadowHit()` vyhodnotí prípadný priesečník s objektom, a pokiaľ je daný objekt pre požiadavku vyhovujúci, vráti sa identifikátor objektu.

Overenie validity prvku nemusí spočívať len v jednoduchom zistení, či sa jedná o požadovaný prvok. V hrách, kde nedochádza ku kolíziám figúr v hernom poli, je pri požiadavke na jedno pole hracej dosky je možnosť, keď vyslaný lúč síce trafil políčko hracej plochy, ktoré je však obsadené inou figúrou absolútne neprijateľná.

5.3.3 Optimalizácie

Efektivita algoritmu vrhania lúčov je vzhľadom na svoju náročnosť vysoko závislá od optimalizácií zavedených do jeho výpočtu. Medzi efektívne spôsoby urýchlenia výpočtu priesečníkov patrí napríklad použitie vhodných stromových štruktúr na organizáciu objektov (*bounding volume hierarchy*) alebo rozdelenie samotného priestoru (*k-d tree*, kde je každý uzol tvorený bodom v k rozmeroch). Jedná sa o pokročilé metódy, ktoré som sa rozhodol pre nízku náročnosť scény neimplementovať.

V prípade ďalšej práce na tomto projekte však bude naprosto nevyhnutne zvoliť a implementovať vhodnú formu optimalizácie.

6 Záver

Cieľom tejto práce bolo prehľadne zhrnúť možnosti zobrazovania obmedzenej scény a vytvoriť aplikáciu, ktorá funguje ako správca modulov jednotlivých doskových hier. Súčasťou je taktiež demonštračný modul s hrou piškvorky pre dvoch hráčov.

Pomocou nadobudnutých znalostí v oblasti vývoja pre platformu Android som navrhol a s využitím knižnice OpenGL ES implementoval interaktívny herný manažér. Jednotlivé hry vystupujú ako samostatné Android aplikácie a do manažéra sú nahrané automaticky po ich inštalácií. Spúšťajú sa skrz jeho rozhranie, nie sú schopné samostatného chodu.

Herná scéna dostupná po spustení hry je zobrazovaná pomocou algoritmu vrhania lúčov, ktorý bol v tejto práci aj bližšie popísaný. Po korektnom ukončení hry a odmietnutí ponuky na repete sa aplikácia vracia späť do manažéra.

Napriek tomu, že je na projekte odvedené nemalé množstvo roboty, vďaka vhodne navrhnutému jadru ostáva široký priestor k ďalšiemu vývoju. Ten je možný plánovať v rámci všetkých častí aplikácie.

Úvodný herný správca by mohol obsahovať možnosť dostupné hry triediť a zobrazovať ako niekoľkoúrovňové grafické menu alebo poskytovať možnosť vyhľadávania podľa názvu či hracej doby. Komplexným rozšírením manažéra by bola podpora komunikácie po sieti, ktorá by umožňovala každému hráčovi hrať na svojom vlastnom zariadení.

V rámci implementácie jednotlivých hier ako modulov je možné rozšíriť protokol komunikácie (samozrejme v kooperácií s implementáciou v grafickom rozhraní hier) o časti podporujúce zavedenie vedenia informácií o herných zdrojoch alebo umožniť použitie herných mechanizmov ako je náhoda vo forme hodu kocky.

Rozšírenia spojené so zobrazovanou scénou použitou metódou sú viazané na optimalizácie tohto procesu a smerom do budúcnosti sú vo vysokej miere odkázané na vývoj hardvéru určeného pre mobilné zariadenia

Kto vie, snáď sa raz dočkáme pešiaka stínajúceho nepriateľskú kráľovnú ako z filmu, a to všetko v scéne vykresľovanej metódou vrhaním svetelného lúča.

Literatúra

[1] Board games: Ancient board games. In: *Wikipedia: the free encyclopedia* [online]. 2014 [cit. 2014-05-12]. Dostupné z: http://en.wikipedia.org/wiki/Board_game

[2] Abstract strategy games and other genres out of scope of IAGO. *International Abstract Games Organization* [online]. [cit. 2014-05-12]. Dostupné z: http://iagoweb.com/wiki/game-genres

[3] Glossary. In: *Board Game Geek* [online]. [cit. 2014-05-12]. Dostupné z: http://www.boardgamegeek.com/wiki/page/Glossary#toc93

[4] Race game. In: *Wikipedia: the free encyclopedia* [online]. 2014 [cit. 2014-05-12]. Dostupné z: http://en.wikipedia.org/wiki/Race_game

[5] SHIRLEY, Peter. *Realistic ray tracing*. Vyd. 2. Massachusetts: Natick, 2003, 225 s. ISBN 15-688-1198-5.

[6] PHONG. Illumination for Computer Generated Pictures. In: [online]. [cit. 2014-05-12]. DOI: 10.1145/360825.360839. Dostupné z: http://www.cs.northwestern.edu/~ago820/cs395/Papers/Phong_1975.pdf

[7] HEROUT, Adam. *Počítačová grafika: Studijní opora*. 2008. Dostupné z: https://www.fit.vutbr.cz/study/courses/PGR/private/PGR-Opora.pdf

[8] OpenGL ES. *Android Developers* [online]. [cit. 2014-05-12]. Dostupné z: https://developer.android.com/guide/topics/graphics/opengl.html

[9] Applying Projection and Camera Views. *Android Developers* [online]. [cit. 2014-05-12]. Dostupné z: https://developer.android.com/training/graphics/opengl/projection.html

[10] Android NDK. *Android Developers* [online]. [cit. 2014-05-12]. Dostupné z: https://developer.android.com/tools/sdk/ndk/index.html

[11] License. *Blender.org* [online]. [cit. 2014-05-12]. Dostupné z: http://www.blender.org/about/license/

[12] Object Files (.obj). *Wavefront Technologies* [online]. [cit. 2014-05-18]. Dostupné z: http://www.martinreddy.net/gfx/3d/OBJ.spec

[13] SIMPSON, Robert J. THE KHRONOS GROUP INC. *The OpenGL® ES Shading Language*. 2009. Dostupné z: http://www.khronos.org/registry/gles/specs/2.0/GLSL_ES_Specification_1.0.17.pdf

[14] Texample2. *Texample2 GitHub* [online]. [cit. 2014-05-18]. Dostupné z: https://github.com/d3kod/Texample2

[15] Defining Shapes. *Android Developers* [online]. [cit. 2014-05-18]. Dostupné z: http://developer.android.com/training/graphics/opengl/shapes.html

[16] Services. *Android Developers* [online]. [cit. 2014-05-18]. Dostupné z: http://developer.android.com/guide/components/services.html

Zoznam príloh

Príloha A

Obsah CD

`/bin/` – preložené spustiteľné súbory

`/doc/` – generovaná Javadoc dokumentácia

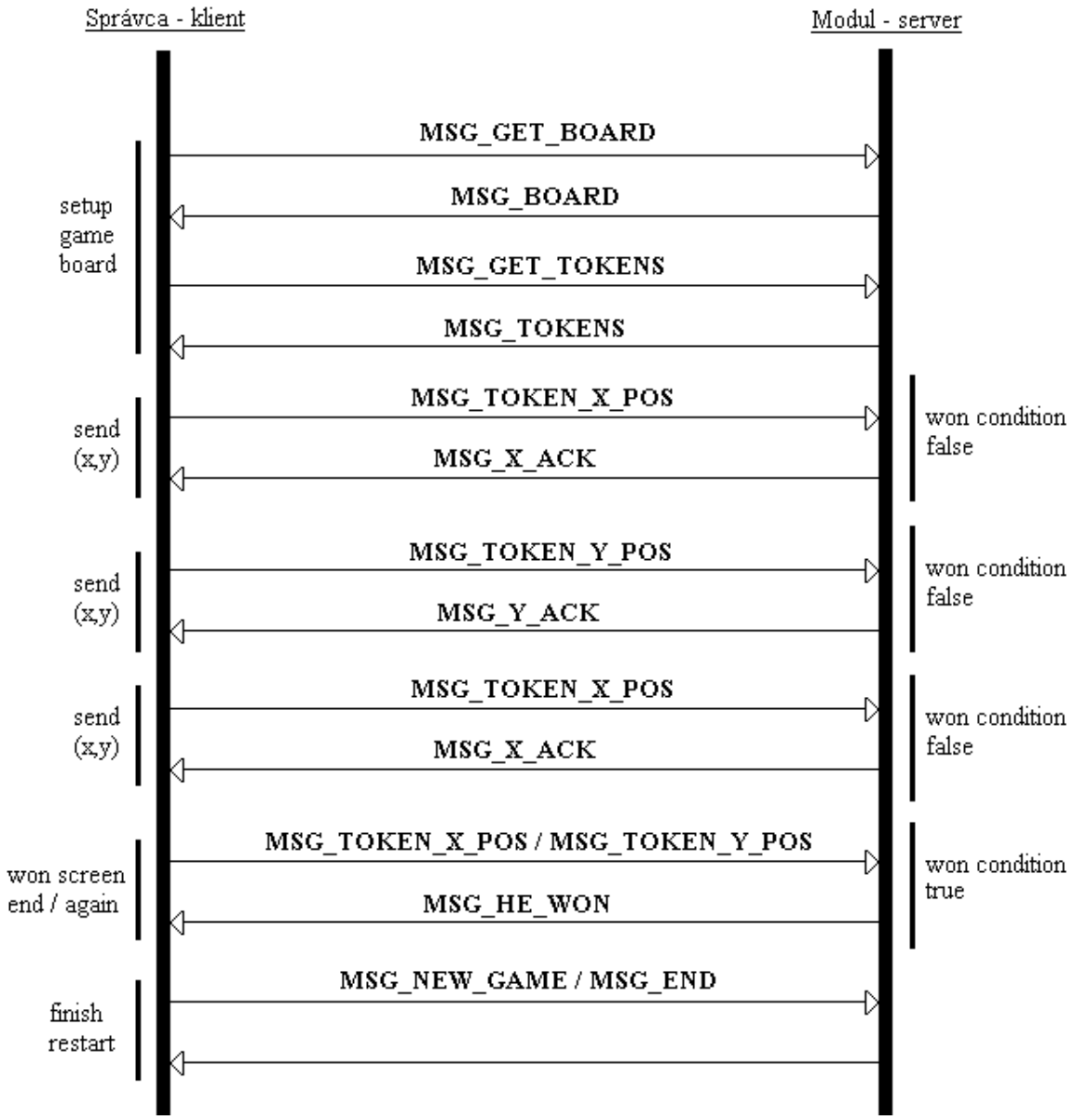
`/poster/` – prezentačný plagát

`/projects/` – obsahuje 2 Android projekty so zdrojovými kódmi (manažér a hra)

`/text/` – dokumenty písomnej správy

`/README` – návod na inštaláciu aplikácií

Príloha B



Príloha A 1 Diagram toku správ pri hre piškvorky