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

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Valter Kasper
AUTHOR

BRNO 2015

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 ROZHRANÍ PRO DESKOVÉ HRY NA PLATFORMĚ ANDROID

GRAPHICAL INTERFACE FOR BOARD GAMES ON ANDROID PLATFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Valter Kasper
AUTHOR

VEDOUCÍ PRÁCE — Ing. Lukáš Polok
SUPERVISOR

BRNO 2015

Abstrakt

Práce prezentuje návrh a implementaci grafického uživatelského rozhraní pro deskové hry. Objektem zkoumání byly deskové hry a grafické efekty pro virtuální scénu deskové hry. Vypracovali jsme návrh uživatelských rozhraní pro různé hry. V rámci práce jsme se seznámili s knihovnou OpenGL ES a jejím použitím na platformě Android. Prozkoumali jsme možnosti implementace v nativním kódu. Vytvořili jsme systém zásuvných modulů, který dovoluje implementovat herní logiku pomocí skriptovacího jazyka a definovat scénu deskové hry pomocí jazyka XML.

Abstract

This work presents a design and implementation of graphic user interfaces for board games. The objects of the investigation were various types of board games and graphic effects for virtual board game scene. We have developed a user interface design for different games. In the following sections the OpenGL ES library and its application to the Android platform are characterized. We explored the possibility of implementation in the native code. We have developed a system for plug-ins that allows implementing game logic in scripting language and defining the virtual scene for board game in XML language.

Klíčová slova

uživatelské rozhraní, zásuvný modul, OpenGL ES, desková hra, Android, NDK, grafické efekty, OpenGL rozšíření, skriptování her

Keywords

user interface, plug-in, OpenGL ES, board game, Android, NDK, graphic effects, OpenGL extensions, game scripting

Citace

Valter Kasper: Grafické rozhranie pre doskové hry na platforme Android, diplomová práce, Brno, FIT VUT v Brně, rok 2015

Grafické rozhranie pre doskové hry na platforme Android

Prehlásenie

Prehlasujem, že som túto diplomovú prácu vypracoval samostatne pod vedením Ing. Lukáša Poloka. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

……
Valter Kasper
25.5.2015

Poďakovanie

V tejto sekcii by som sa chcel poďakovať vedúcemu práce a tým, ktorí mi poskytli odbornú pomoc.

© Valter Kasper, 2015
Tato práce vznikla jako školní dílo na Vysokém učení technickém v Brně, Fakultě informačních technologií. Práce je chráněna autorským zákonem a její užití bez udělení oprávnění autorem je nezákonné, s výjimkou zákonem definovaných případů.

Obsah

1 Úvod ... 1
2 Doskové hry ... 2
2.1 Príklady stolných hier ... 3
3 Návrh herných rozhraní ... 8
3.1 Herné rozhranie hry Vlk a ovce ... 8
3.2 Herné rozhranie hry Othello ... 10
4 OpenGL ES na platforme Android ... 12
4.1 Knižnica OpenGL ES ... 12
4.2 Android a OpenGL ... 12
4.3 Použitie natívneho kódu ... 13
4.4 OpenGL rozšírenia na platforme Android ... 14
5 Grafické efekty ... 16
5.1 Rozostrenie obrázkov ... 16
5.2 Vykresľovanie do textúry ... 16
5.3 Celoplošné grafické efekty ... 17
5.4 Grafické efekty na vybraných herných prvkoch ... 20
5.5 Výber objektu v 3D scéne ... 21
6 Návrh systému zásuvných modulov ... 23
6.1 Existujúce východiská ... 24
6.2 Návrh komponent ... 24
6.3 Definícia hernej dosky ... 25
6.4 Rozhranie herná logika – BoardEngine ... 28
6.5 Podpora pre vlastnú hernú logiku ... 29
6.6 Podpora pre tvorbu GUI ... 31
7 Implementácia ... 32
7.1 Architektúra systému ... 32
7.2 Zásuvný modul ... 33
7.3 Podporná knižnica pre prácu s OpenGL ... 41
7.4 Grafický engine ... 44
7.5 Implementovaná dosková hra ... 53
8 Zhodnotenie dosiahnutých výsledkov ... 56
8.1 Porovnanie s inými prístupmi ... 56
8.2 Test výkonu ... 57
8.3 Budúci vývoj ... 58
9 Záver ... 60
Literatúra ... 61
Popis skratiek ... 63
Zoznam príloh ... 64

1 Úvod

Podstatou stolných hier sú ich pravidlá, ktoré dovoľujú hráčom aplikovať stratégie tak, aby dosiahli požadovaný cieľ. Hranie hry je o to zaujímavejšie, ak je hraná proti zdatnému protihráčovi. Tvorba takej hry pre stolné počítače alebo aj mobilné zariadenia, by mala hlavne pozostávať z implementácie hernej logiky a umelej inteligencie, no nie je tomu celkom tak. Ak má byť hra nie len funkčná, ale aj príjemne vyzerať, tak je nutné, venovať množstvo času vývoju grafického užívateľského rozhrania. Naša práca, sa preto zameria na vytvorenie 3D užívateľského rozhrania pre istý typ doskových hier, ktoré zefektívni ich tvorbu. Toto užívateľské rozhranie bude určené pre mobily a tablety platformy Android a bude ho možné použiť implementáciou zásuvného modulu.

V sekcii číslo 2 klasifikujeme rôzne typy stolných hier a zároveň popíšeme niekoľko doskových hier, aby sme sa lepšie zoznámili s problematikou. Nasleduje sekcia 3, kde navrhneme užívateľské rozhranie a spôsob ovládania a pre dve hry. Potom sekcia 4 priblíži knižnicu OpenGL ES so zameraním na platformu Android a možnosť implementácie v natívnom kóde. V sekcii 5 budú objektom skúmania grafické efekty, ktoré by bolo vhodné použiť v našom frameworku. V pod sekcii 5.1 sa bližšie pozrieme na techniku rozostrenia obrázkov a v 5.2 na techniku vykresľovania do textúry. Oddiel číslo 6. zachytí návrh systému zásuvných modulov. Ukáže spôsob, akým by mohol klient nadefinovať hernú dosku a implementovať hernú logiku. Potom bude nasledovať kapitola 7 venovaná implementácii, ktorá sa bude venovať popisu tvorby a použitiu nášho systému. V podsekcií 7.5 si ukážeme, ako vznikal zásuvný modul – dosková hra s použitím nášho systému.

2 Doskové hry

Doskové stolné hry by sme mohli charakterizovať ako podmnožinu stolných hier, od ktorých sa líšia tým, že dej doskovej hry prebieha na špeciálnej hernej ploche – hernom pláne – prostredníctvom herných kameňov. Hranie doskovej hry sleduje určitý cieľ, ktorý sa dá dosiahnuť aplikovaním stratégie alebo vplyvom náhody (napr. hod kockou), často však obomi. Naša práca sa zameriava na klasické hry, preto sme zvolili delenie doskových hier podľa [1]. Delenie je na základe herného deja a úlohy hráča:

- **Strategické hry**. Hlavným rysom sú boje dvoch a viac armád, kde sa kamene zajímajú. Zajaté kamene sa väčšinou vyhadzujú z hernej plochy. Dajú sa rozdeliť podľa spôsobu, akým je zajímanie vykonávané – preskakovaním (Dáma), obkľúčenie (Vlk a ovce), vytvorenie formácie a iné.
- **Závodné hry**. Majú úplne odlišný cieľ ako strategické hry. Ich cieľom je dostať figúrky do cieľa skôr ako protivník. Rýchlosť figúrky často závisí od náhody – hod kockou. Pri pretekoch môže dochádzať ku konfrontácii. Pri tomto type hier je typický vyšší počet hráčov. Do tejto kategórie patrí Backgammon, Človeče nehnevaj sa alebo Monopoly.
- **Pozičné hry**. Hráči sa snažia klásť kamene tak, aby vytvorili istú zostavu a tak splnili úlohy. Sem patria Piškôrky a Mlyn.
- **Pátracie hry**. Hráči riešia istý problém alebo niečo hľadajú. Patria sem napríklad detektívne hry alebo potápači.

Hranie hier ma vplyv na trpezlivosť, predstavivosť a rozvoj hráča. Pri hraní sa dajú ľahko odhaliť ľudské charaktery protihráčov. Podľa archeologických nálezov boli prvé hry hrané na dolnom toku rieky Níl, v štátoch Sumeru – medzi riekami Eufrat a Tigris, v severnej Indii a Číne [2]. Zmena epochy v dejinách často prestavovala zmenu formy a obsahu hier.

Ako sa spoločnosť sa menila na expanzívnu a otrokársku, tak sa začínali dobyvačné výpravy. Výsledkom boli nové územia a zajatci. V tomto období vznikajú hry ako Dáma a Go. V rodine hier so základom v Dáme, vyhráva ten hráč, ktorý zničí alebo zajme celú nepriateľskú armádu. Hry odvodené od Go tiež napodobňujú vojnu, ale inak. Podstatou je obkľúčenie vojska veľkým počtom nepriateľov, pričom je dôležitá aj obrana.

Feudalizmus zaviedol hierarchiu spoločenských vzťahov, čo sa odrazilo aj na dobových hrách. Vrchol spoločenskej pyramídy predstavoval vládca – cisár, kráľ či knieža. Príkladom hry reflektujúcej danú situáciu je šach. V šachu ide o zbavenie moci vládcu, čím vojna končí. S nástupom kapitalizmu, doby kariéry, prichádza hra monopoly. Monopoly popisujú realitu, v ktorej vznikli. Obsahujú herné prvky ako fiktívne peniaze alebo kapitál.

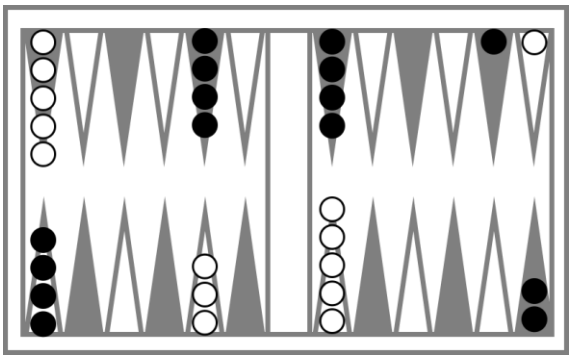
Nová generácia hier, podľa [2], bude používať meniace sa herné dosky, pričom sa budú používať elektronické zariadenia alebo v hrách bude veľký vplyv náhody čoby obraz osudu človeka.

2.1 Príklady stolných hier

Predtým, ako navrhneme rozhranie stolných hier, ktoré bude možné vytvoriť vo frameworku, prezentujeme niekoľko rôznych stolných hier. Potom sa budú jednoduchšie analyzovať herné prvky. Pri jednotlivých hrách zachytíme princíp herných pravidiel a popíšeme hernú dosku.

2.1.1 Backgammon

Hra Backgammon sa často hravá v stredozemí. Je to národná hra Gruzíncov, Arménov, Arabov a Turkov, ďalej je hrávaná v štátoch bývalej Juhoslávie a Bulharsku [2]. Backgammon má zvláštnu hernú dosku. Pozostáva z dvoch strán, s dvanástimi predlženými klinmi, ktoré predstavujú herné polia (Obrázok 1). Hru tvorí 30 kameňov, z ktorých je 15 bielych a 15 čiernych. Hráči sa

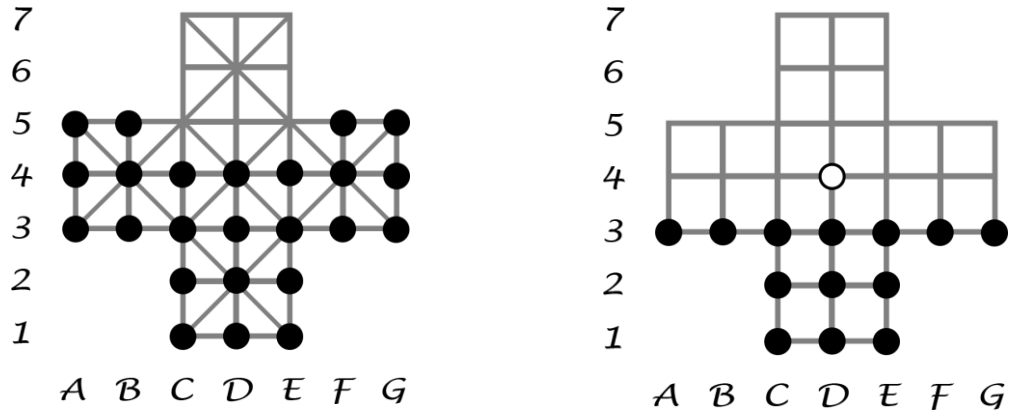


Obrázok 1 Príklad rozohranej partie hry Backgammon.

striedajú, pričom každý hádže v jednom kole dva krát kockou. Hodnoty na kockách sa sčítajú a výsledné číslo predstavuje posun herných kameňov po herných poliach. Hráč môže v jednom kole posúvať jedným alebo dvomi kameňmi. Pri hode rovnakého čísla, dubletu, hráč posúva figúrky o dvojnásobný počet políčok. Hra sa končí, keď hráč vyvedie všetky svoje kamene von z hracej plochy.

2.1.2 Vlk a ovce

Hra vznikla na Islande, približne v 14. storočí, a následne sa rozšírila po celej Európe. Identické hry ako Vlk a ovce vznikli aj v Ázii [2]. Herná doska pozostáva z 33 polí. Kamene sa ukladajú na priesečníky čiar. Ako z názvu vyplýva, v hre jestvujú dva typy figúrok. Ovce majú 13 čiernych kameňov a vlk má jeden biely kameň (Obrázok 2). Ťahy prebiehajú striedavo, pričom ovce sa pohybujú len dopredu a do strán a vlk sa pohybuje aj dozadu. Vlk berie, vyhadzuje z hernej plochy, ovce v priamom smere preskočením. Ovce sa do hry ďalej nevracajú. Brať je možné aj viac oviec naraz. Hráč hrajúci s ovcami vyhráva, keď obkľúči vlka tak, že nemôže vykonať ďalší ťah. Naopak vlk vyhráva, keď počet oviec klesne natoľko, že ho nemôžu obkľúčiť.

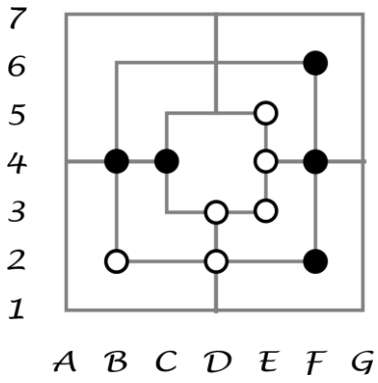


Obrázok 2 Vľavo je možné vidieť začiatok hry pevnosť. Na voľné polia sa umiestnia dvaja obrancovia a hrať je možné po diagonálach. Vpravo je počiatočné rozloženie figúrok hry Vlk a ovce.

Existuje množstvo variácii základnej hry. Napríklad hra sa dá hrať s rozšírením, keď je umožnený pohyb po diagonálach. Inou modifikáciu pôvodnej hry je hra Pevnosť. Pevnosť pozostáva z 24 útočníkov a dvoch obrancov. Na začiatku sú obrancovia postavení na ľubovoľné voľné herné pole. Útočníci vyhrávajú, ak obkľúčia obrancov alebo ak obsadia 9 pozícií pevnosti. Tento variant tiež umožňuje použiť diagonály, inak sa hrá ako pôvodná hra.

2.1.3 Mlyn

Hra Mlyn je jedna z najstarších hier vôbec. Archeologické a etnologické nálezy ukázali, že bola známa po celom svete [2]. Hracia doska pozostáva z 18 kameňov (9 čiernych a 9 bielych). Patrí do kategórie hier, kde sa kamene ukladajú na priesečníky čiar.



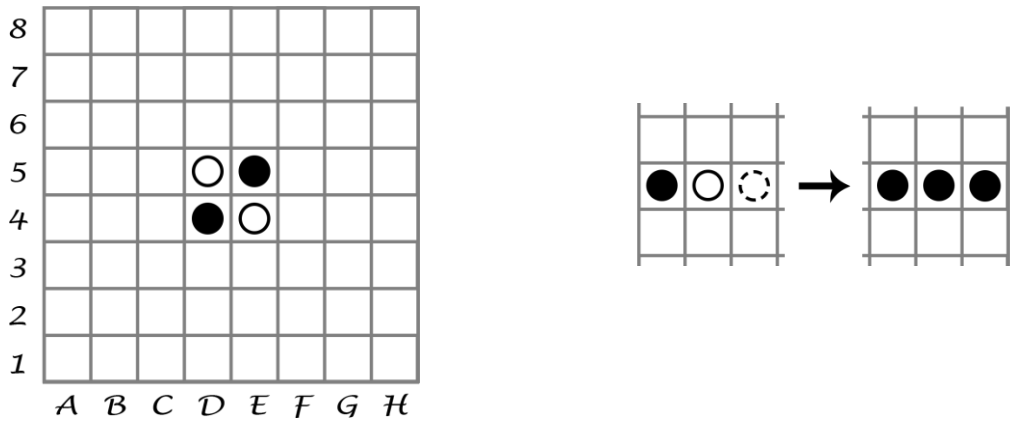
Obrázok 3 Ukážka z partie hry Mlyn. Biely hráč je na rade, pričom má vytvorený mlyn. Teraz môže vyhodiť čiernu figúrku.

Hra je rozdelená do troch etáp. V prvej etape hráči na striedačku rozostavujú kamene po hracej ploche. V druhej etape prebieha samotná hra, keď sa kamene pohybujú vždy o jedno políčko po hranách medzi políčkami. Tretia etapa začína, keď aspoň jedenému hráčovi ostali iba tri

kamene. V tomto prípade môže položiť tento hráč kameň kdekoľvek. Hra končí, keď jednému hráčovi ostanú iba dva kamene. Ťah v ľubovoľnej etape je nasledujúci: Ak má hráč tri vlastné kamene vedľa seba (spojené čiarou), takzvaný *mlyn*, tak zoberie súperovi jeden kameň (Obrázok 3). Ak sa mu podarí postaviť dva mlyny, tak zoberie dva kamene.

2.1.4 Othello

Othello je hra s jednoduchými pravidlami. Plocha na hranie pozostáva z dvojfarebných kameňov (Obrázok 4 vľavo). Hráči ťahajú na striedačku, pričom sú dovolené len ťahy, kde sa berie súperov kameň. Ak hráč nemôže vykonať ťah, tak je na rade protihráč. Branie prebieha cez obstúpenie protihráčovho kameňa s tým, že obstúpený kameň zmení farbu (Obrázok 4 vpravo).

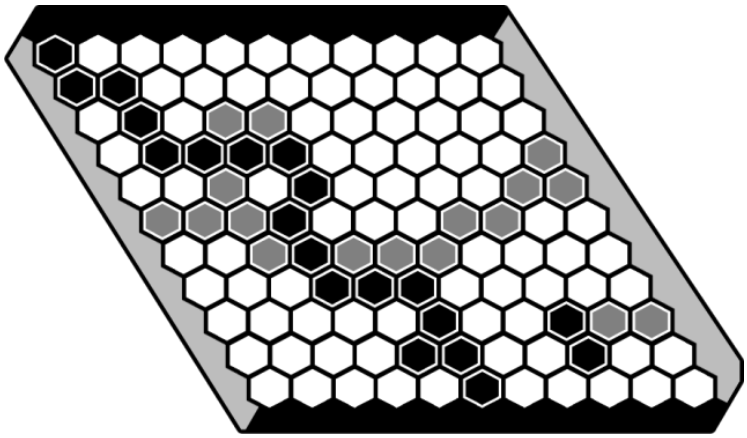


Obrázok 4 Vľavo: úvodné rozostavenie kameňov v hre Othello. Vpravo: obstúpenie protihráča. Čierny hráč obstúpil bieleho, tým sa biely kameň zmenil na čierny a pribudol ďalší čierny kameň.

Tým postupne, na hracej ploche pribúdajú kamene. Je možne zobrať viac ako jeden kameň. Partia končí, keď je zaplnené celé pole alebo ak neexistuje ďalší ťah. Vyhráva hráč s viac kameňmi.

2.1.5 Hex

Moderná stolná hra vyvinutá Dánom Pietom Heinom v roku 1942. Herná doska pozostáva z 122 kameňov, kde 61 je bielych a 61 čiernych políčok, ktoré majú šesťuholníkový tvar. Hráči sa striedajú v každom kole a ukladajú kamene na dosku. Po uložení kameňa sa s ním už nehýbe a kamene sa neberú. Partiu vyhráva ten hráč, ktorý urobí reťaz svojich kameňov medzi protiľahlými stranami (Obrázok 5).

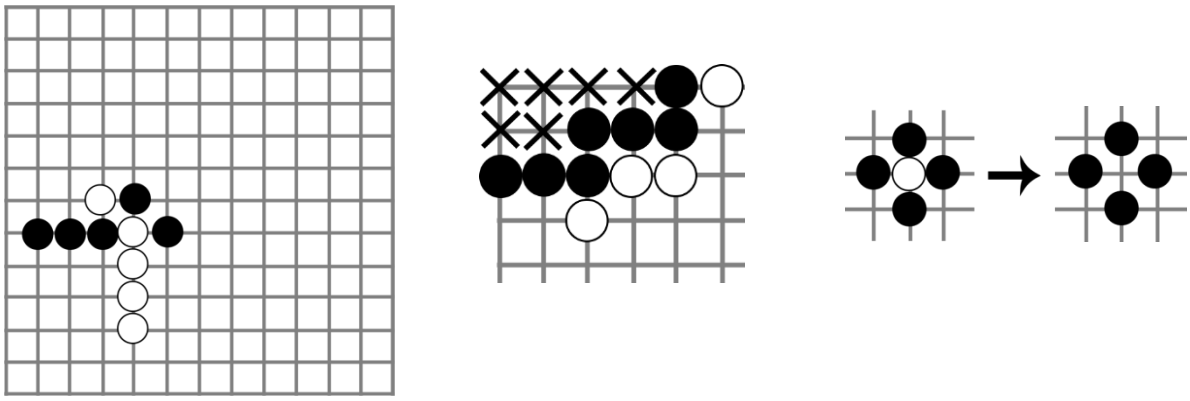


Obrázok 5 Koniec hry. Hráč s čiernymi figúrkami spojil protiľahlé strany.

2.1.6 Go

Jedná sa o najstaršiu dodnes hranú stolnú hru. Herné pravidlá tejto hry sú dostatočne jednoduché a preto sa v priebehu vekov nemenili. Svoje korene má v starovekej Číne pred 4000 – 5000 rokmi. V siedmom storočí sa Go dostalo do Japonska, kde sa stalo národnou hrou [2].

Herné kamene sa striedavo ukladajú na polia, ktoré sú tvorené priesečníkmi čiar na doske s rozmermi 19x19 polí (začiatočníci môžu použiť rozmery 9x9 alebo 13x13). S položenými kameňmi sa ďalej nehýbe, ale za určitých okolností sa vyberajú. Cieľom je, za prvé, obkľúčiť čo najviac nepriateľských kameňov, za druhé, zahradiť čo najväčšie územie. Obkľúčením súperových kameňov (Obrázok 6 vpravo) sa tieto kamene dajú do misky. Obkľúčenie musí byť napevno, bez voľných polí, a zároveň obkľúčenie stačí po čiarach.



Obrázok 6 Vľavo je príklad rozohranej partie hry Go. V strede možno vidieť zahradenie územia hráčom s čiernymi figúrkami. Vpravo je obkľúčenie bieleho kameňa čiernymi. Obkľúčený kameň sa presunie do misky

Zahradeným územím sa myslí územie, ktoré je obohnané kameňmi hráča, od hrany k hrane hracej plochy (Obrázok 6 v strede). Hra končí, keď hráči nemôžu alebo nechcú vykonať ďalší ťah. V tom prípade obaja hráči povedia *pas*. Vyhráva hráč, ktorý ma viac zahradených políčok a zajatých

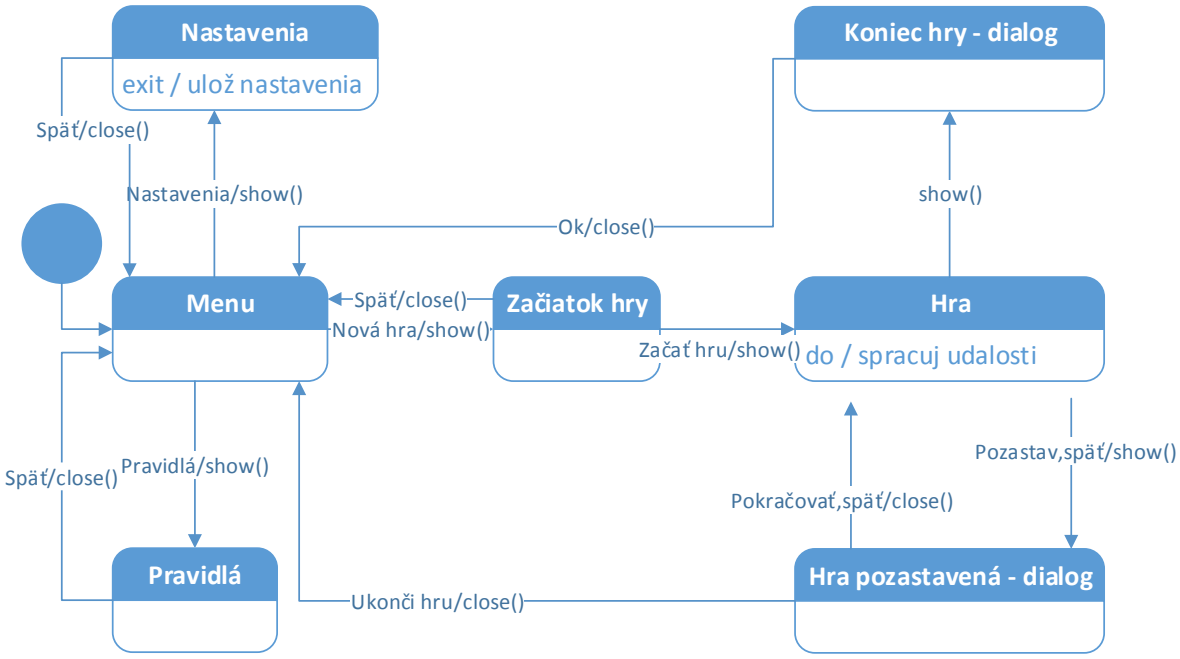
nepriateľských kameňov. V partiách hráčov s rovnakou úrovňou, je výhoda prvého ťahu kompenzovaná takzvaným *komi*, ktoré si biely hráč pripočíta na konci partie. Hodnota *komi* je závislá na pravidlách, často má hodnotu 5,5.

3 Návrh herných rozhraní

Návrhy herných rozhraní pomôžu určiť požiadavky na framework. Vymedzia spôsoby zobrazenia hernej plochy, spôsoby interakcii a typy udalostí, ktoré môžu nastať a na ktoré potreba reagovať. UI užívateľovi dovolí efektívne ovládanie aplikácie a súčasne aplikácia poskytne informácie, ktoré užívateľovi uľahčia rozhodovanie. V rámci návrhu herných rozhraní špecifikujeme pre každú hru:

- Herné obrazovky – budú špecifikované formou obrázku a textu.
- Prechody medzi obrazovkami – bude použitý UML stavový diagram.
- Ovládanie hry – zahrňuje manipuláciu hernej dosky ako celku a manipuláciu jednotlivých herných prvkov.
- Popis herných prvkov – hernej dosky a figúrok.

Rozhodli sme sa popísať užívateľské rozhranie hier Othello a Vlk a ovce hry. Sú to klasické hry, ktoré sa hrajú na poliach so štvorcovými políčkami. V hre Othello sa figúrky kladú na stred políčka a v hre Vlk a ovce, na priesečníky políčok.



Obrázok 7 Prechody medzi hernými obrazovkami.

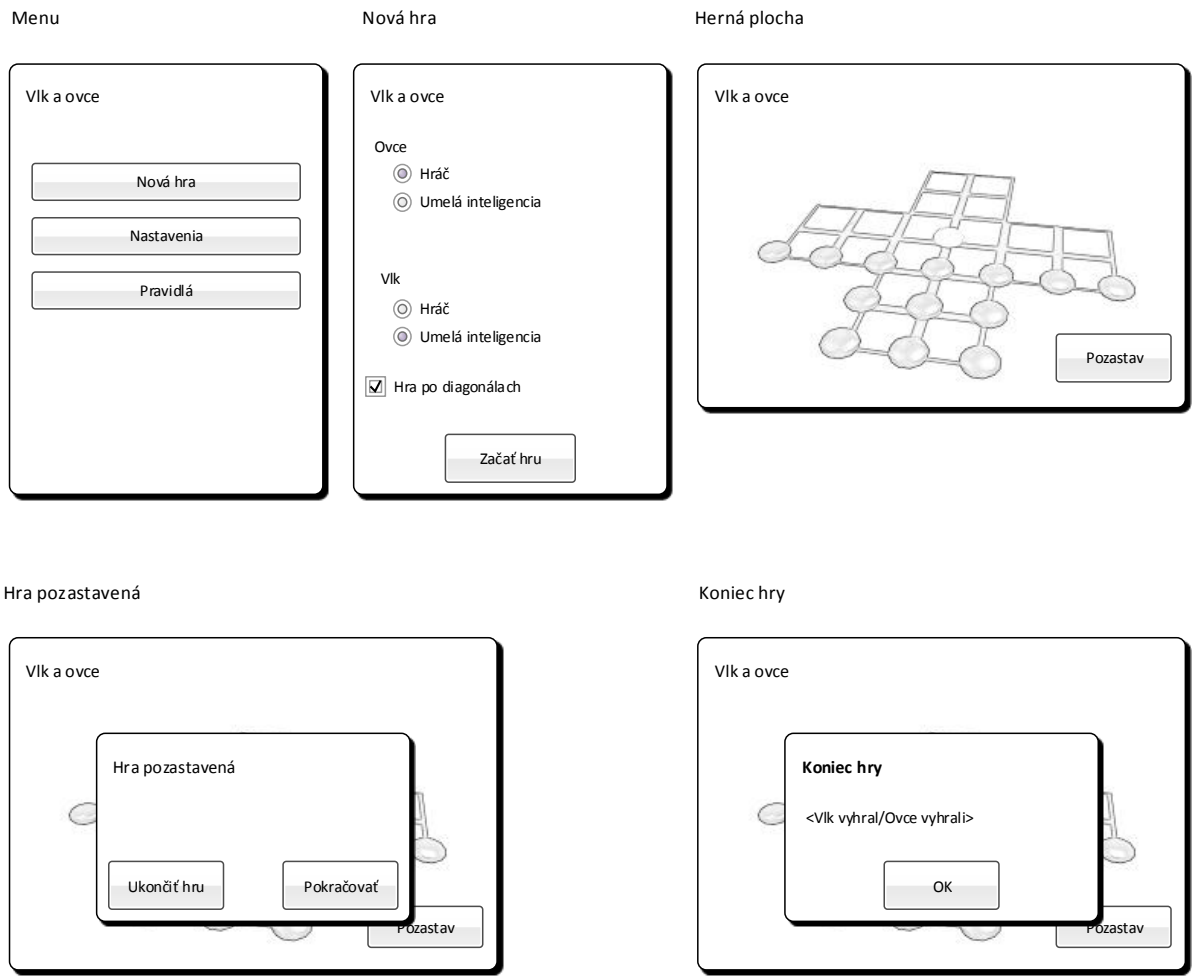
3.1 Herné rozhranie hry Vlk a ovce

V nasledujúcich častiach si navrhneme a popíšeme herné rozhranie hry Vlk a ovce.

3.1.1 Herné obrazovky

Aplikácia pozostáva z niekoľkých obrazoviek. Medzi obrazovkami sa prechádza spôsobom, ako je vyobrazené v UML stavovom diagrame (Obrázok 7). Vzhľad niektorých obrazoviek je ilustrovaný na obrázku (Obrázok 8). Textový popis obrazoviek je nasledujúci:

- **Menu**. Počiatočná obrazovka hry. Obsahuje položky *Nová hra*, *Nastavenia* a *Pravidlá*.
- **Nová hra**. Dovoľuje nastaviť novú hru. Minimálne je to voľba strany (ovce, vlk), voľba protihráča (hráč alebo umelá inteligencia) a dovolenie hry po diagonálach.
- **Pravidlá**. Bude obsahovať dokument s pravidlami a spôsobom ovládania hry.
- **Hra pozastavená**. Dočasné pozastavenie rozohranej hry. Z tejto obrazovky sa dá hra ukončiť a vrátiť sa do menu.
- **Nastavenia**. Bude obsahovať užívateľom definované preferencie. Napríklad grafické a zvukové nastavenia.
- **Herná plocha**. Vykresľuje 3D užívateľské rozhranie, zobrazujúce priebeh hry. Reaguje na dotykové udalosti od užívateľa. Herná doska bude zasadená do 3D statického prostredia,



Obrázok 8 Niektoré obrazovky užívateľského rozhrania.

na ktoré sú aplikované grafické efekty a postprocesing. Ukazovatele o stave hry nie sú potrebné – hra je dostatočne jednoduchá.

- **Koniec hry**. Hra samotná detekuje víťaza a následne zobrazí víťaznú obrazovku s informáciou o víťazovi. Je automaticky zobrazená, keď nie je možné vykonať ďalší ťah.

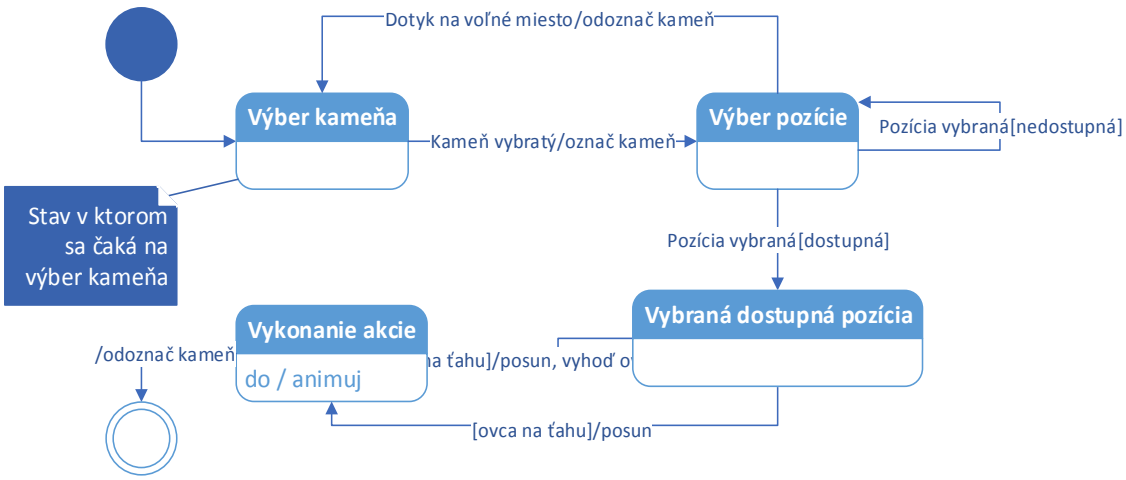
3.1.2 Popis herných prvkov

Kamene budú mať tvar oválu, líšiť sa budú farbou. Budú meniť farby, podľa stavu v ktorom sa nachádzajú. Herná doska bude mať tvar ako na obrázku na strane 4 vpravo.

3.1.3 Ovládanie hry

Pri ovládaní hernej plochy ako celku, budú použité dotykové gestá. Medzi akcie, spustené gestami, patrí otáčanie, približovanie a pohybovanie do strán. Ak užívateľ bude chcieť priblížiť dosku, tak priloží dva prsty na obrazovku a roztiahne ich. Ak ju bude chcieť otočiť, tak dva prsty otočí okolo stredu. Ak chce pohnúť dosku do strany alebo hore a dole, tak prstom pohne v danom smere.

Ťah hráča začína výberom kameňa (Obrázok 9). Výber kameňa (ovce alebo vlka) prebieha jednoducho tak, že sa hráč dotkne políčka, na ktorom stojí. Daná figúrka sa zvýrazní. Následne sa vykoná ťah dotykom na zvolené políčko. Ak ťah nie je platný, tak sa políčko sfarbí červenou farbou. Inak, sa vykoná animovaný pohyb na dané políčko. Tak isto vyhodenie ovce z hracieho poľa je animované.

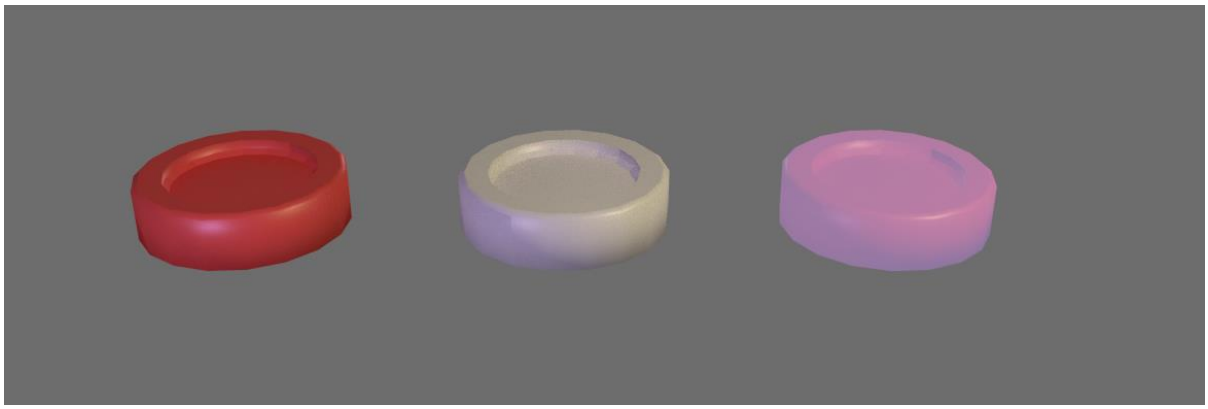


Obrázok 9 Postup krokov pri ťahu, znázornený ako UML stavový diagram.

3.2 Herné rozhranie hry Othello

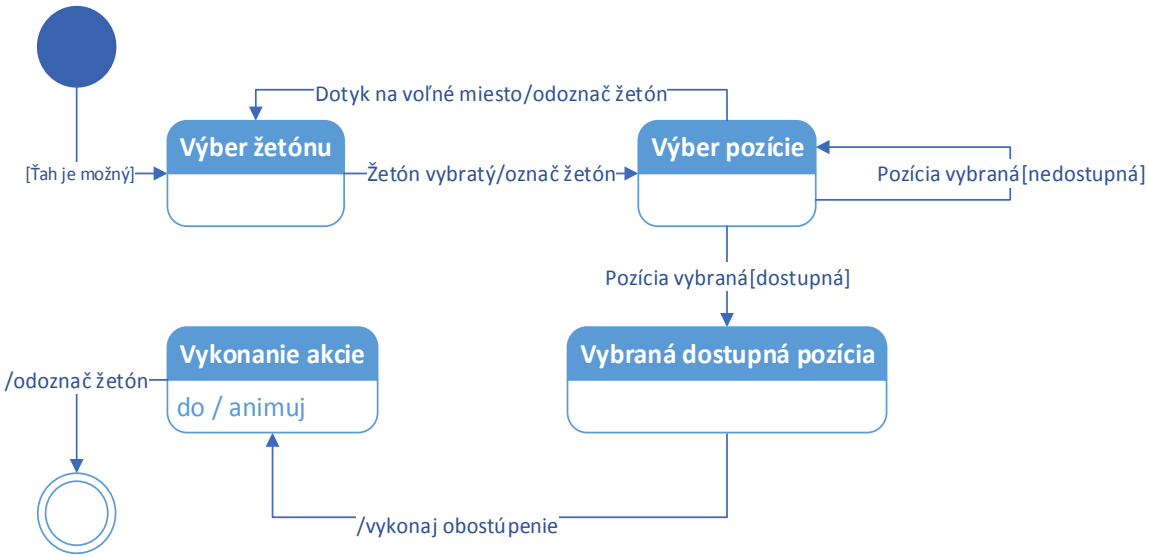
Hra Othello obsahuje figúrky v dvoch farbách. Figúrky budú zobrazené ako žetóny (Obrázok 10). Herná doska má veľkosť 8x8 políčok. Ťah figúrky prebieha podobne ako v hre Vlk a ovce. Najprv sa vyberie žetón a potom cieľová pozícia (Obrázok 11). Herná plocha sa dá otáčať,

približovať posúvať zo strany na stranu cez klasické dotykové gestá. Obrazovka by mala zobrazovať počet figúrok každého hráča a tlačidlo pre návrat do menu. Po detekcií konca hry (je zaplnené celé pole alebo neexistuje ďalší ťah) sa zobrazí víťazná obrazovka.



Obrázok 10 Možný vzhľad figúrok v hre Othello. Zľava doprava: Žetóny prvého a druhého hráča a označený žetón.

Herná doska bude imitáciou reálnej drevenej dosky. Bude pokrytá textúrou dreva. Medzi grafické efekty patrí odraz žetónov na hracej ploche a dynamické tiene.



Obrázok 11 Ťah v hre Othello. Animovaním je myslené postupné obstúpenie žetónu a postupná zmena farby.

4 OpenGL ES na platforme Android

Aktuálna kapitola pojednáva o knižnici OpenGL na platforme Android. Pokúsi sa priblížiť spôsob práce s OpenGL na tejto platforme. Súčasne si priblížime *Native Development Kit* (ďalej NDK), často používaným na prácu s interaktívnou grafikou na platforme Android.

4.1 Knižnica OpenGL ES

Sekcia, ktorú práve čítate stručne predstaví knižnicu OpenGL ES. Zhrnie princípy a popíše verzie knižnice. *Open Graphic Library for Embedded Systems* – skrátene OpenGL ES – je nízkoúrovňové API, umožňujúce prepojenie softwaru s grafický hardwarom zariadenia. OpenGL ES je podmnožinou OpenGL pre stolné počítače. Programové rozhranie je prenositeľné na množstvo zariadení od herných konzol, cez mobilné zariadenia, po dopravné prostriedky. Knižnica je preto vhodná na vytvorenie prenositeľného grafického kódu, pre rôzne platformy.

OpenGL ES existuje v niekoľkých verziách. Podľa čísla verzie sa líši spôsob programovania. Jestvujú dve hlavné vydania: Verzia OpenGL ES 1.1, má najväčšiu podporu na medzi zariadeniami, no používa zastarané API s fixným grafickým reťazcom. Z toho vyplýva pevná množina operácii na transformáciu a vykresľovanie geometrie. Naproti tomu verzia OpenGL ES 2.0 umožňuje programovať niektoré stupne grafického reťazca. To dovoľuje vytváranie programov, ktoré sú spustené na grafickom hardwari. Pre programovanie grafických komponent je použitý jazyk GLSL. Výsledkom môžu byť komplexnejšie grafické efekty. Existujú dva základné programovateľné stupne. Jeden pre manipuláciu s vrcholmi – *vertex shader* a druhý pre manipuláciu s fragmentmi – *fragment shader*. Poslednou verziou knižnice je verzia 3.x, ktorá oproti predchádzajúcej verzií neprináša také zásadné zmeny. Medzi hlavné prídavky patrí podpora pre komprimované textúry (bez potreby rozšírení), *geometry shader* a *tesselation shader*.

4.2 Android a OpenGL

Android pripúšťa použitie všetkých verzií OpenGL ES. No podpora konkrétnej OpenGL verzie závisí od podpory grafického hardwaru daného zariadenia. V súčasnosti, takmer všetky zariadenia podporujú verziu 2.0 a rýchlo sa zväčšuje podiel zariadení s podporou verzie 3.x.

Pre použitie knižnice, je najprv potrebné naviazať OpenGL kontext na Android okno aplikácie. Na tento účel sa používa pomocná knižnica *Embedded System Graphic,* skrátene EGL. OpenGL sa môže programovať cez manažovaný (na platforme Android sa tým myslí Java programovací jazyk). Android framework poskytuje jednu základnú triedu a jedno rozhranie:

- Trieda **`GLSurfaceView`** umožňuje manipuláciu a vykresľovanie objektov cez OpenGL API. Rozšírením tejto triedy je možné reagovať na udalosti dotyku. Inštancií tejto triedy sa predáva objekt, ktorý implementuje rozhranie `GLSurfaceView.Renderer`.
- Rozhranie **`GLSurfaceView.Renderer`** vyžaduje implementáciu operácií: `onSurfaceCreated`, `onDraw` a `onSurfaceChanged`. Renderer je zodpovedný za volanie OpenGL a vykreslenie snímkou.

Typickou požiadavkou pri vývoji grafických aplikácii je načítanie externých zdrojov ako sú obrázky či zdrojové kódy shaderov. Na to sa dá použiť trieda `AssetManager`, ktorá dovoľuje prístup k zdrojom v zložke `/assets`, ktoré sú zabalené do APK archívu pri zložení aplikácie. Také zdroje sú používané ako binárne dáta. Pristupuje sa k ním relatívne, voči zložke `/assets`, cez ich názov. Pri programovaní na Androide sa používajú aj zdroje, ktoré sú umiestnené v priečinku `/res`. Tento typ zdrojov nie je dostupný cez NDK. Ďalším spôsobom prístupu je cez externé úložisko. V tomto prípade sa dá použiť štandardné C/C++ rozhranie na manipuláciu so súbormi. Programovanie OpenGL môže prebiehať kompletne z manažovaného kódu alebo sa volania OpenGL presunú do natívneho kódu.

4.3 Použitie natívneho kódu

Hry alebo multimediálne aplikácie, potrebujú rýchlu odozvu a veľa systémových prostriedkov. Na to je vhodné použiť NDK – *Native Development Kit*. Je to súbor nástrojov, ktorý poskytuje kompilačný reťazec (*compiler chain*), dokumentáciu a niekoľko dôležitých API. Takým API môže byť napríklad OpenGL ES, OpenSL ES alebo OpenMAX AL. Ďalším dôvodom použitia NDK, okrem efektívneho nakladania so systémovými zdrojmi, je prenositeľnosť C/C++ kódu.

V predošlých vydaniach NDK bola slabá podpora pre výnimky, RTTI (*Run-Time Type Information*) a STL (*Standart Template Library*). Teraz sa toto obmedzenie dá obísť jednoduchou zmenou konfigurácie v prekladovom súbore `Application.mk`.

Pri použití NDK pri vývoji aplikácie, vo väčšine prípadov, je požadované prepojenie s Java časťou aplikácie. Pre toto prepojenie je využívané *Java Native Interface* (ďalej JNI), ktoré dovoľuje kooperáciu medzi Javou a C/C++ kódom. Príkladom použitia je vytvorenie UI v Jave a volanie natívnych operácii implementovaných v C/C++. Postup pre použitie natívneho kódu je nasledujúci:

1. Deklarovanie natívnych operácii na Java strane aplikácie.
2. Použitie nástroja javah, ktorý vygeneruje hlavičkové C/C++ súbory z Java tried. Nástroj javah sa požíva na skompilované .class súbory. Výsledkom je špecifická menná konvencia v hlavičkových súboroch.
3. Definovanie natívneho kódu.

Na Java strane sa natívne metódy deklarujú cez kľúčové slovo `native`. Potom sa načítajú dynamické knižnice, ktoré implementujú natívne metódy cez volanie príkazu `System.loadLibrary()` alebo `Runtime.loadLibrary()`. Preklad natívnych aplikácii na platforme Android sa vykonáva cez špecifický `Makefile`, nazvaný `Android.mk`. Preklad sa spustí príkazom `ndk-build`.

Iný spôsob použitia natívneho kódu, je tzv. natívna aktivita. Dovoľuje implementáciu *callback* funkcii na udalosti životného cyklu aplikácie v natívnom kóde. Preto sa dedí z triedy `NativeActivity`, v ktorej sa preťažujú metódy ako `onCreate`, `onPause` alebo `onResume`, tak ako v normálnej Android aktivite.

4.4 OpenGL rozšírenia na platforme Android

Mechanizmus rozšírení v OpenGL dovoľuje, aby vývojári hardwaru pridávali nové prvky vytváraním rozšírení, ktoré môžu použiť softwarový vývojári. Nové prvky sú pridávané nad rámec špecifikácie OpenGL.

Používajú sa preto, aby vývojári mali prístup k novým vlastnostiam, ktoré nie sú dostupné v štandarde. Mnoho rozšírení poskytuje vlastnosti, ktoré sú dostupné vo vyššej verzii štandardu. Každé rozšírenie má unikátny názov a mnohé sú závislé na výrobcovi grafického hardwaru.

Naša aplikácia bude používať OpenGL ES vo verzii 2. Preto nás budú zaujímať najmä rozšírenia, ktorých funkcionalita nie je obsiahnutá v tomto štandarde. Z pohľadu našej aplikácie by bolo vhodné, aby sme použili tieto rozšírenia resp. operácie, ktoré ponúkajú:

ETC1 textúry

V našom prípade síce nebudeme používať vo veľkej miere obrázky na textúrovanie objektov, ale je to predpoklad, že herné obrazovky budú obsahovať tlačidlá a iné grafické prvky, ktoré vyžadujú textúry vo vysokom rozlíšení. Bolo by asi vhodné použiť komprimované textúry.

Rozšírenie `OES_compressed_ETC1_RGB8_texture` umožňuje použitie komprimovaných ETC1 textúr. Textúry je vhodné komprimovať, pretože zaberajú množstvo miesta v pamäti grafickej karty. Normálne komprimované formáty ako .jpg či .png sú po nahratí do grafickej karty rozbalené do svojej nekomprimovanej podoby. Na rozdiel od toho, existujú kompresné formáty, ktoré zostanú komprimované aj po nahratí do grafickej karty. Takým jedným, široko podporovaným kompresným formátom je ETC1. Jeho hlavnou nevýhodou je, že nepodporuje priamo alfa kanál a poskytuje iba stratovú kompresiu. Používa sa volaním funkcie `glCompressedTexImage2D()`.

Okrem toho existuje formát ETC2, podporujúci aj alfa kanál. Je dostupný na všetkých zariadeniach s OpenGL 3.0. Existujú aj ďalšie formáty, ale tie sú závislé na výrobcovi grafického hardwaru.

Instanced rendering

V mnohých scénach sa vykresľuje veľký počet inštancii jedného objektu s mierne odlišnými parametrami, napr. rôzne rozmery, iná základná farba či pozícia. V našom prípade budeme vykresľovať typicky veľké množstvo rovnakých figúrok, líšiace sa len polohou a farbou. Aby sme také objekty vykreslili, tak pre každý objekt nastavíme rôzne parametre v podobe uniformných premenných a následne ho vykreslíme.

Lepší prístup je *instanced rendering* – inštančné vykresľovanie, ktorý spočíva v tom, že v jednom príkaze vykreslíme viacero inštancií rovnakého modelu naraz, pričom poskytneme každej inštancii vlastné parametre. Existujú dva hlavné prístupy. V prvom dostane vertex shader potrebné parametre ako atribút. V tomto prípade sa atribút nemení pri každom vrchole, ale napríklad pri každej inštancií. Dosiahnuť tohto efektu je možné pomocou rozšírenia `NV_instanced_arrays`. Druhý prístup spolieha na vstavanú premennú `gl_InstanceID`, ktorá je dostupná vo vertex shadery. Táto premenná začína počítať od nuly a zvyšuje sa po každom vykreslení inštancie. Hodnota premennej `gl_InstanceID` sa môže použiť ako index do uniformného poľa. Rozšírenie má názov `NV_draw_instanced`. Tento spôsob vykresľovania je volaný z aplikácie príkazom `glDrawElementsInstanced()`. Na rozdiel od funkcie `glDrawElements()`, prijíma aj počet vykresľovaným inštancií. Je nutné podotknúť, že spomínané rozšírenia sú dostupné, len na niektorých zariadeniach s grafickým akcelerátorom od firmy NVIDIA.

Float textúry

Niektoré grafické efekty produkujú v medzi kroku hodnoty typu float, ktoré sa zapisujú do textúry framebuffra. OpenGL vo verzií 2.0, ale nepodporuje float textúry. Riešením môže byť zabalenie float hodnoty do RGBA komponent textúry manuálne v shadery alebo použitie rozšírenia `EXT_color_buffer_half_float`. Spomínané rozšírenie umožňuje pripojiť 16 bitovú float textúru ako cieľ vykreslenia do framebufferu. Toto rozšírenie by mohlo nájsť uplatnenie, pri ukladaní informácie o hĺbke v algoritme tieňových máp alebo pri výpočte úrovne rozostrenia v bloom efekte.

5 Grafické efekty

V tejto kapitole sa zameriame na grafické efekty. Uvedené efekty sú vybrané vzhľadom k zameraniu našej scény a jej použitiu – vykreslenie hracej plochy a figúrok. Okrem konkrétnych grafických efektov popíšeme niektoré spoločné prístupy používané v mnohých efektoch. Spomenieme rozmazanie textúr alebo vykresľovanie do textúry.

5.1 Rozostrenie obrázkov

Rozostrenie obrázkov je v zásade jednoduché. Sčítajú sa okolité pixely, spriemerujú sa a máme požadovanú hodnotu. No sú tu rôzne cesty, ako to vykonať s rôznym vizuálom, kvalitou a efektivitou. Filtrovanie textúry je súčasťou mnohých algoritmov v počítačovej grafike. Medzi inými sa dá použiť v efektoch Hĺbka ostrosti alebo Bloom efekt

Základný prístup spočíva v aplikovaní Gausovho filtru. Volá sa Gausov, pretože na rozmazanie používa Gausovú krivku. Bežne sa aplikuje tak, že sa vytvorí konvolučný filter, teda sa použije 2D matica s predpočítanými hodnotami Gausovej funkcie. Tieto hodnoty sa použijú ako váhy pri sčítavaní susedných pixelov. Tým vznikne nová hodnota pixelu.

Je možné nahliadnuť, že náročnosť výpočtu pre jeden pixel je $O(n^2)$. Kde n je veľkosť jadra – šírka alebo výška matice. Vyriešiť tento problém umožňuje separovateľnosť 2D filtra na dva 1D filtre. Tak filter môžeme rozdeliť do vertikálneho a horizontálneho priechodu. Výsledná náročnosť výpočtu je $O(n)$ na pixel a rovnaká obrazová kvalita. Ešte jedným vylepšením je interpolácia váh Gausovho filtra, čím sa zníži počet vzoriek. Je to vhodné hlavne pre veľké jadrá [3].

Gausov filter je ešte stále náročný na výpočet. Ukazuje sa, že ak sa rozmazávaný obrázok zmenší, následne sa aplikuje filter a opäť sa zväčší, tak výsledok má porovnateľnú kvalitu ako filtrovanie cez plné rozlíšenie. Optimalizácia spočíva v tom, že sa jednak spracováva menšia plocha (napr. 4 násobné zmenšenie pri polovičných stranách) a súčasne je možné zmenšiť jadro, pretože sa vzorkuje v podstate väčšia plocha naraz.

Na záver spomeňme **Kawase blur fiter** [4]. Je to viac priechodový prístup, ktorý v každom priechode aplikuje malé množstvo rozostrenia tak, že sa spriemeruje 16 pixelov. Výstup z priechodu sa privádza na vstup ďalšieho. Výsledok je aproximáciou použitia Gausovho filtra. Kawase filter je výkonnejší ako optimalizovaný Gausov filter, hlavne pre väčšie jadrá [3].

5.2 Vykresľovanie do textúry

Grafické efekty spravidla pozostávajú z viacerých krokov. Bežnou technikou pre implementáciu grafických efektov je použitie viacerých priechodov, pričom informácie medzi

priechodmi sú predávané v podobe textúr. Vykresľovaný obraz putuje do pamäti snímku – *framebuffer*. Obsah framebuffer-u nemusí nutne odpovedať obrazovke zariadenia, môže sa jednať tzv. off-screen framebuffer. Framebuffer sa skladá z niekoľkých častí, je to akási schránka na obrazy. Týmito časťami sú *Color buffer, Depth buffer a Stencil buffer*. V OpenGL sa používa na nastavenie aktívneho framebufferuu príkaz `glBindFramebuffer`, pričom pri vykresľovaní môže byť aktívny iba jeden framebuffer. Spomínané časti framebuffera sa označujú ako ciele – *render targets*. K týmto cieľom sa pristupuje ako ku textúram. V rámci jedného priechodu je možné na s knižnicou OpenGL ES vo verzií 2.0 vykresľovať iba do jedného cieľa.

Samozrejme, vykresľovanie mimo obrazovku zariadenia kladie výpočtové nároky. Existujú spôsoby, ako sa s tým vyrovnať. Jeden prístup spočíva v obmedzení rozlíšenia textúry a farebnej hĺbky, čím sa zníži pamäťová náročnosť. Druhý prístup je vykresľovanie textúry len na vyžiadanie.

5.3 Celoplošné grafické efekty

Atraktivitu scény, ako celku, možno zvýšiť aplikovaním efektov. V tejto kapitole si prestavíme niekoľko efektov, ktoré sú uplatnené na celú scénu.

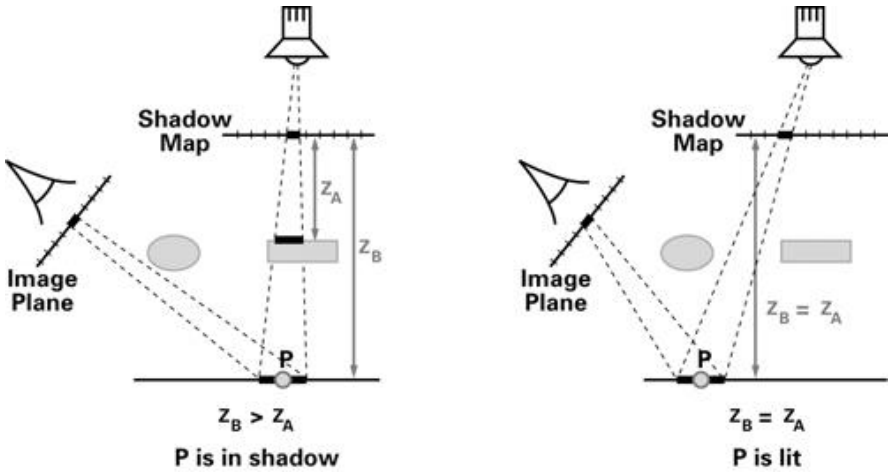
5.3.1 Tieňové mapy

Tiene poskytujú vizuálnu informáciu o relatívnej vzdialenosti objektov, pozícii svetla, tvare objektu, ktorý vrhá tieň a dodávajú realizmus do scény. Preto si na začiatok predstavíme *Shadow maps* – tieňové mapy [5]. Tieňové mapy patria medzi efekty, ktoré používajú viac priechodov a používajú projekčné mapovanie textúr. Táto metóda pracuje aj pre scény s komplikovanou geometrou, ktorá vrhá vlastné tiene. V prvom priechode sa vypočíta tieňová mapa z pohľadu svetla, na čo je potrebné vypočítať pohľadovú maticu. Preto je nutný smer a poloha svetla. Pre bodové svetlá sa aplikuje perspektívna projekcia, pre smerové svetlá projekcia ortonormálna. Cieľová projekcia bude vykreslená do framebuferu s aktivovaným depth buffrom a bude obsahovať informáciu o hĺbke, teda vzdialenosti od svetla.

Druhý priechod používa získanú tieňovú mapu na určenie, ktorá časť scény sa nachádza v tieni a ktorá nie. Táto mapa, je v podobe textúry privedená na vstup fragment shaderu. Scénu teraz vykreslíme z pohľadu oka. Fragmenty sa prevedú do súradnicového systému svetla a porovná sa z súradnica s hodnotou v tieňovej mape. Ako textúrovacie súradnice sa použijú x a y súradnice transformovanej pozície.

Výhodou algoritmu je jednoduchá implementácia základného algoritmu. Na druhú stranu, nie je jednoduché dosiahnuť dostatočnú kvalitu. Problémom sú artefakty spojené s obmedzenou presnosťou použitých dátových typov. A tak vznikajú zúbky na hranici tieňa, pretože susedné body sa mapujú na rovnaké texely v textúre. Je možné navýšiť rozlíšenie tieňovej mapy a tým predísť

problému so vzorkovaním. Iným riešením je metóda, postavená na vzorkovaní viac hodnôt z okolia pôvodného vzorku a ich spriemerovanie, čím sa rozmažú okraje tieňa. Tento princíp používa metóda **PCF** (Percentage-Closer Filterng) [6]. Získava porovnanie bodov z okolia – *n* pravdivostných hodnôt. Následne sa filtruje výsledná hodnota (všetky vzorky majú rovnakú váhu). Výsledkom je obmedzený počet hodnôt intenzít tieňa. Obísť sa to dá upravením váh podľa vzdialenosti vzorkou. Vzdialenosť v ktorej sa vzorkuje môže byť konštantná alebo nepravidelná. Pri nepravidelnej vzdialenosti vzorkou od stredu sa často používa **Poissonov disk** [7], teda body sú rovnomerne rozložené v jednotkovej kružnici a vzdialenosť bodov nie je nižšia ako definovaná hodnota. V porovnaní s pravidelnou PCF mriežkou dáva Poissonov disk vzorky, ktoré vedú k uspokojivejším výsledkom.

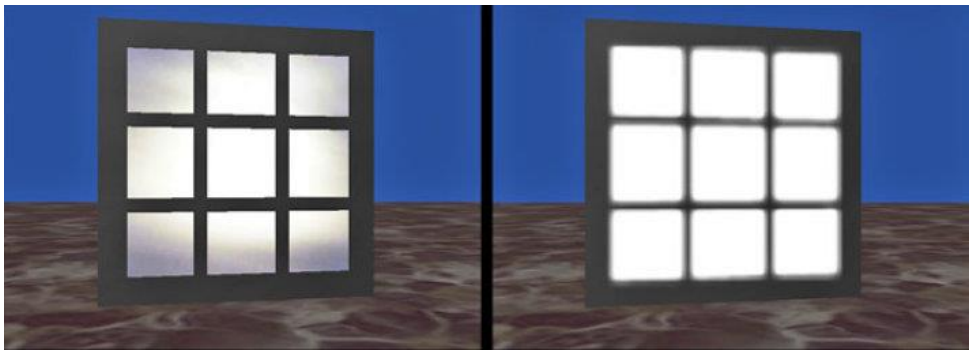


Obrázok 12 Princíp tieňových máp. Naľavo je vyobrazený prípad, keď z súradnica bodu, je väčšia ako hodnota v textúre mapy. Napravo sa hodnoty z súradnice a hodnoty z mapy rovnajú. Tým dochádza k z-fightu a artefaktu zvanému akné. Je to možné jednoducho odstrániť. K hodnote textury sa pripočíta malé číslo – *bias*. (Obrázok prevzatý z [8]).

5.3.2 Bloom efekt

Niekedy je žiaduce, vyjadriť vysokú intenzitu svetla. Vysokú intenzitu svetla môžu vyžarovať zdroje svetla alebo lesklé povrchy, a tým okolo seba vytvoriť viditeľnú auru. Tento jav sa dá vyjadriť pomocou bloom efektu [9], ktorý je možné vygenerovať pomocou shaderov a viacerých priechodov: V prvom priechode vykreslíme bloom textúru a scénu do štandardného framebufferu. V prípade bloom textúry vykreslíme iba časti, ktoré presahujú istý prah intenzity svetla. Rozlíšenie bloom textúry môže byť nižšie, pretože bude použité rozmazanie.

Ďalej rozmažeme bloom textúru, napr. pomocou Gausovho filtra, ako je spomenuté v sekcii 5.1. Bežne sa používa jadro o veľkosti 5x5. Nakoniec sa rozmazaná textúra primieša do štandardného framebufferu.

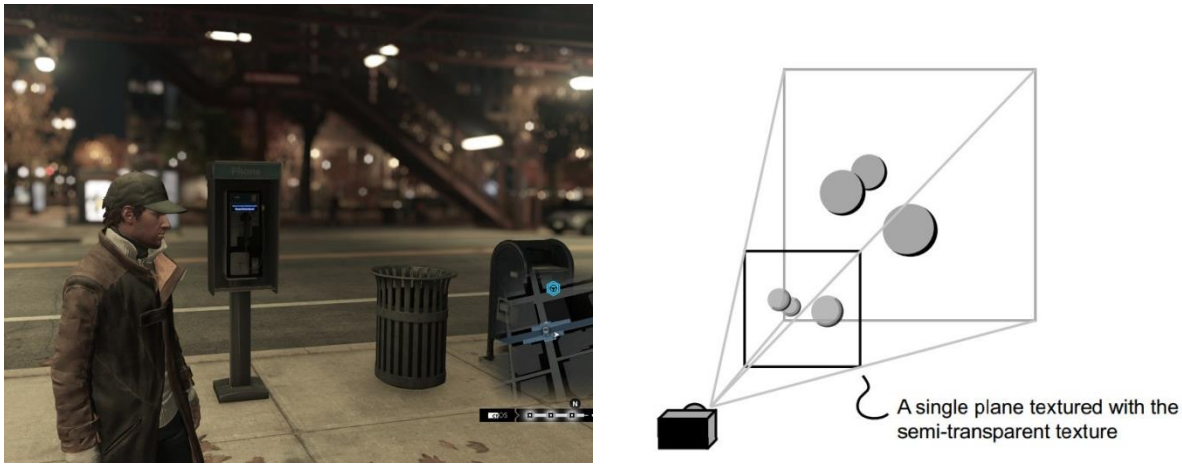


Obrázok 13 Ukážka scény bez a s bloom efektom. Obrázok z [9].

5.3.3 Hĺbka ostrosti

Hĺbka ostrosti (angl. *Depth of field*) je súčasťou videnia človeka alebo optiky fotoaparátu, či kamery. Označuje rozsah vzdialenosti, v ktorom sa javia objekty ostré [10]. Vo fotoaparátoch majú na hĺbku ostrosti vplyv tri faktory: ohnisková vzdialenosť objektívu, clona a vzdialenosť objektu. V počítačovej grafike sa vykresľujú snímky ako cez dierkovú kameru [11]. To spôsobuje. že scéna je úplne ostrá. Existuje niekoľko spôsobov, ako efekt hĺbky ostrosti implementovať aj do virtuálnej scény. Najpresnejším prístupom je simulácia prechodu svetla zo scény na priemetňu, pričom svetlo prechádza cez sústavu šošoviek. Iný spôsob je, vykresliť scénu niekoľko krát, pričom pozícia kamery sa o niečo mení. Výsledné nasnímané obrazy sa potom spoja, pred vykreslením finálneho obrazu.

My sa zameriame na techniku, ktorá vykreslí scénu dva krát – do textúry a na obrazovku [10]. Scéna vykreslená do textúry bude rozmazaná. Pri mixovaní s obrazovkou sa použije priehľadnosť



Obrázok 14 Vľavo je ukážka efektu hĺbky ostrosti v hre Watch dogs. Rozmazaním pozadia sa upriamila pozornosť na postavu v popredí. Vpravo je spôsob vykreslenia efektu hĺbky ostrosti. V popredí, na úrovni orezovej platne je polygón, na ktorý sa vykreslí rozmazaná textúra. Obrázok prevzatý z [10]

tejto textúry. Táto hodnota priehľadnosti odpovedá tomu, ako je objekt v scéne zaostrený. Hodnota rovná 1 znamená úplné zaostrenie. Úroveň zaostrenia sa počíta ako vzdialenosť od zaostrovacieho dobu. V našom prípade budeme pracovať so zaostrovacou platňou rovnobežnou s prednou orezovou platňou (*cliping plane*). V okolí platne je priestor, ktorý je zaostrený, a zaostrenie postupne klesá. Šírka tohto priestoru sa dá nastaviť parametrom, ktorý významom odpovedá clone.

Problémom je vykresľovanie priehľadných objektov Zatiaľ čo priesvitný objekt je zaostrený, tak objekt za ním nemusí byť a naopak. Možnosťou je vykresliť priesvitné objekty až po aplikácii efektu, čoho následkom je, že objekt je vždy ostrý, ale objekty za ním budú rozmazané.

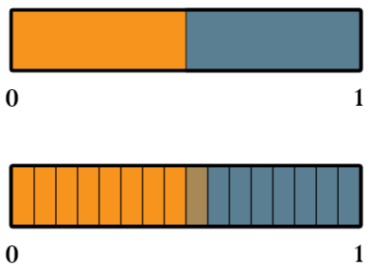
5.4 Grafické efekty na vybraných herných prvkoch

Častým cieľom počítačovej grafiky je dosiahnutie realistického vzhľadu scény. Vizuálne príťažlivou alternatívou môže byť tzv. nerealistická grafika, ktorej zámerom je istá štylizácia scény. Snaží sa zaujať inak, než realistickým vzhľadom. Tohto môžeme dosiahnuť aplikovaním špecifického vzhľadu vykresľovaných objektov, napríklad použitím charakteristického tieňovania povrchu objektu.

5.4.1 Toon shading

Bežná technika v počítačovej grafike je *toon shading* – kreslený vzhľad objektov. Toon shading je tieňovacia technika, keď odtiene farieb sú ostro ohraničené [13]. Jeden z možných prístupov je nasledujúci: Difúzna farba je prevedená na iba dva tóny, základný a tmavý. Spekulárna farba je reprezentovaná iba jedným, jasným tónom. Nakoniec pre dodanie kresleného vzhľadu, sa pridá čierna silueta.

Intenzita difúzneho osvetlenia je počítaná skalárnym súčinom $\vec{N}.\vec{L}$. Kde $\vec{N}$ je normála povrchu a $\vec{L}$ je smer svetla. Výsledná hodnota je sklár označujúci svetlé, resp. tmavé časti objektu. Jedným spôsobom, ako previesť túto súvislú hodnotu na ostré úrovne môže byť použitie 1D textúry, ktorá má

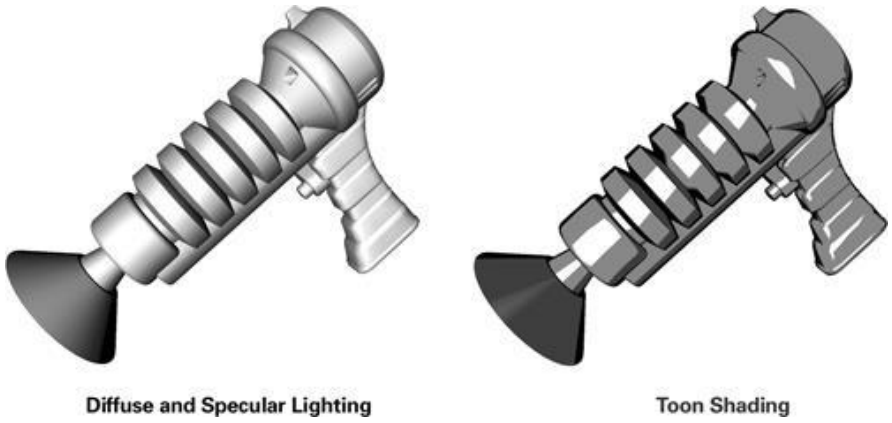


Obrázok 15 Podoba 1D textúr pre toon shading. Horný obrázok ukazuje textúru s dvomi hodnotami. Spodný obsahuje prechodovýodtieň.

n textelov, takže výsledný objekt bude mať n odtieňov. Hodnota intenzity svetla bude

použitá ako textúrovacia súradnica. Použitie striktných hraníc vedie k aliazingu. Na pomoc prichádza väčšia, upravená 1D textúra, ktorá zakóduje plynulejšie prechody (Obrázok 15). Zároveň sa zapne bilineárne filtrovanie textúr, ktoré vyhladí prechody automaticky.

Posledným krokom je stvárnenie siluety objektu. Cieľom tejto techniky je nájsť pixely, ktoré sa nachádzajú na okraji objektu a vyfarbiť ich farbou siluety. K tomu sa dá použiť výraz $\vec{N}.\vec{V}$, kde $\vec{V}$ je smer kamery. Výsledkom výrazu je informácia o postavení povrchu vzhľadom k oku. Negatívne hodnoty patria plochám odvráteným od pozorovateľa a hodnoty blízke 0 reprezentujú hraničné časti – hľadanú siluetu.



Obrázok 16 Vľavo je objekt vykreslený pomocou Phongovho stieňovania. Na pravej strane je objekt vykreslený iba pomocou niekoľkých tónov. Je možné vidieť dva tóny difúznej zložky, jeden tón spekulárnej zložky a zvýraznenú siluetu objektu. Obrázok prevzatý z [8].

5.5 Výber objektu v 3D scéne

Výber objektu je proces, keď sa snažíme označiť geometrický objekt v 3D scéne z jeho 2D projekcie na obrazovke zariadenia tým, že je naň klikneme myšou alebo sa ho dotkneme prstom [14]. Výber dotykom je najintuitívnejší spôsob interakcie v 3D scéne. Prístupov pre výber objektu je niekoľko. V nasledujúcich sekciách si predstavíme niekoľko z nich.

5.5.1 Ray casting

Princípom R*ay casting* metódy je, že sa vyšle lúč z pozície kamery do bodu v scéne, ktorý sa premieta na obrazovku v mieste dotyku. Lúč sa prevedie do svetových koordinát a vyhľadajú sa všetky objekty, ktoré sa pretínajú s lúčom. Výber objektu vyžaduje prechádzanie všetkých objektov, aby sme zistili možný prienik. U každého objektu sa zisťuje existencia prieniku s jeho trojuholníkom. Aby sme tomu predišli, možno vytvoriť hierarchickú štruktúru grafu scény. Test prieniku sa potom propaguje do potomkov uzlu len v tom prípade, keď existuje prienik s ohraničujúcim objemom[1]. Okrem samotnej informácie, že došlo k prieniku, môžeme dostáť aj informácie o bode prieniku,

[1] Angl. bounding volume

normálovom vektore prieniku a vlastnostiach povrchu objektu v bode prieniku [14]. Ray casting metóda je relatívne rýchla a nepotrebuje ďalšie vykresľovanie priechody. Okrem výberu objektu je technika Ray casting užitočná pri detekcií kolízii.

5.5.2 Color picking

Metóda *Color picking* – výber podľa farby – je jednoduchá a ľahko použiteľná metóda výberu objektu. Základnou ideou je vykresliť scénu ako zvyčajne, ale jednotlivým objektom sa priradí unikátna farba. Následne sa zistí farba na ktorú bolo kliknuté (pomocou funkcie `glReadPixels`) a prevedie sa na odpovedajúci objekt. Po vybraní objektu sa scéna vykreslí znova, so správnymi materiálmi. K správnemu fungovaniu algoritmu je nutné dodať dve funkcie, ktoré prevedú objekt na farbu a farbu na objekt. Nevýhodou je samozrejme potreba ďalšieho priechodu, čo môže spôsobiť pokles *fps*[2], či neexistencia ďalších informácii o bode výberu.

[2] fps – (angl. frames per second) počet snímkov za sekundu

6 Návrh systému zásuvných modulov

V nasledujúcej časti zhrnieme požiadavky na systém zásuvných modulov. Potom v stručnosti popíšeme, ako by mohol byť definovaný herný modul. V ďalších kapitolách sa budeme bližšie venovať jednotlivým prvkom systému. Ako bolo povedané, hlavným výstupom práce je framework. Jedná sa o framework pre efektívne vytvorenie stolnej hry. Okrem iného hra obsahuje figúrky, ktoré bude možné vyberať a premiestňovať. Takým pohybom môže byť vyhodenie, vytvorenie alebo označenie figúrky. Keďže hra bude spustená na zariadení s dotykovou obrazovkou, tak je dôležité aby tomu bolo prispôsobené aj ovládanie. Hernú plochu by malo byť možne otáčať, posúvať a približovať pomocou gest. Na každú akciu by mal hráč dostať spätnú väzbu. To znamená zvýraznenie herného prvou po dotyku alebo inej vykonanej akcii. Hráč by mal mať jasnú predstavu o tom, čo môže s hernou plochou robiť.

Klient bude môcť, vhodným spôsobom nadefinovať hernú plochu a hernú logiku. Bude mu umožnený import modelov a textúr bežným spôsobom. Ďalej nadefinuje reakcie na udalosti, ako výber herného prvku. Systém poskytne množinu grafických efektov aplikovateľných na jednotlivé figúrky alebo na plochu ako celok.

Systém zásuvných modulov pre doskové hry, by mal klinetom umožňovať definovanie hernej dosky a figúrok. Ďalej umožní naprogramovať hernú logiku, ktorá bude plne pod kontrolou klienta. Táto herná logika môže obsahovať implementované pravidlá doskovej hry, prípadne umelú inteligenciu. Herný engine by mal do zásuvného modulu propagovať udalosti na hernej ploche. Herná logika bude reagovať, cez pevne dané rozhranie, vykonaním akcii nad hernou doskou.

Vzhľadom k požiadavkám, sme navrhli systém, ktorý bude dostatočne univerzálny, aby bolo možné, nad ním vytvoriť množstvo doskových hier. Množina typov doskových, ktoré sa budú dať implementovať, je ale obmedzená. Patria sem hlavne hry pozičné a strategické. Ďalej to musia byť hry, ktoré kladú figúrky na štvorcovú sieť. Do tohto typu hier patrí Go, Dáma, Šachy, Vlk a ovce, Mlyn a iné.

Herná doska sa bude deklarovať cez konfiguračný súbor. Konfiguračný súbor je vo formáte XML, ako je na platforme Android zvykom. Medzi inými budú konfigurovateľné tieto položky: názov modulu implementujúci hernú logiku, rozmer dosky, geometria dosky (súbor), material dosky, typy figúrok, použité grafické efekty. Bližší popis položiek bude nasledovať v ďalších častiach.

Herná logika bude musieť implementovala požadované rozhranie, cez ktoré bude dostávať informácie o zmene stavu životného cyklu a udalostiach na hernej ploche, ktoré sú relevantné k logike hry – kliknutie na políčko a kliknutie na voľnú plochu. Ostatné udalosti sú spracované interne v rámci herného enginu. Herná logika po prijatí udalosti vykoná akcie nad hernou plochou. Môže sa jednať o posun figúrky na políčko, označenie figúrky, označenie políčka alebo dotazy nad

stavom hernej plochy. Snahou pri návrhu bolo, aby sa klient mohol zamerať na logiku hry a na manipuláciu s hernou doskou. Práca s OpenGL ostáva plne v kompetencii herného enginu.

Pojem	Vysvetlenie
Klient	Účastník, ktorý používa framework – vytvára hru. Používa definované API, programuje hernú logiku, vytvára hernú dosku atď.
Hráč	Používateľ výsledného produktu – hry.
Hracia plocha	Časť scény, obsahujúca stolovú hru s políčkami a figúrkami.
Herná doska	Obrazec, na ktorom prebieha dej doskovej hry. Časť hracej plochy dovoľujúca interakciu hráča a hernej logiky. Synonymum pre herný plán.
Políčko	Miesto na hernom pláne, kde sa zbiehajú alebo pretínajú línie. Atomická časť hernej dosky.
Herná logika	Časť aplikácie vytvorenej klientom, implementujúca pravidlá konkrétnej hry.
Zásuvný modul	Časť aplikácie vytvorená klientom. Pokrýva definovanie hernej dosky a hernej logiky.

Tabuľka 1 Slovník doménových pojmov.

6.1 Existujúce východiská

Mobilné platformy ponúkajú, v rámci svojich obchodov, množstvo doskových hier. Značná časť z nich je v 2D zobrazení. Vhodnou voľbou, pre takýto typ zobrazenia je API na zobrazovanie a animáciu 2D prvkov na `Canvas`[3], ktoré ponúka množinu metód, ako `drawBitmap`, `drawRect` alebo `drawText`. Alternatívou môže byť použitie webových technológii a vykresľovať hraciu plochu na objekt typu `WebView`. V tomto prípade, by hracia plocha bola zobrazená pomocou HTML a kaskádových štýlov a herná logiky by mohla byť napísaná v jazyku JavaScript. Doskové hry zasadené do 3D scény, v zásade, používajú priamo grafickú knižnicu OpenGL a DirectX alebo existujúci herný engine.

6.2 Návrh komponent

Náš systém sa skladá z niekoľkých základných komponent. Na UML diagrame tried, na obrázku v prílohe na strane 65, je možno vidieť ich prepojenie.

- **`GameLogic`** – Jedná sa o konkrétnu implementáciu hry. Herná doska bola deklarovaná cez XML. Môže komunikovať iba s naším systémom. Poskytuje rozhranie implementovaním niekoľkých funkcii a pracuje s herným enginom.

[3] http://developer.android.com/reference/android/graphics/Canvas.html

- **`BoardEngine`** – Vysokoúrovňový komponent, ktorý predstavuje jadro nášho frameworku a dal by sa nazvať aj vykresľovacím enginom. Skladá sa z dvoch pod komponent `BoardRenderer` a `BoardController`.
- **`BoardRenderer`** – Zabezpečuje vykreslenie scény aplikáciu efektov, načítanie geometrie a textúr z externých súborov. Udržuje si zoznam figúrok, ktoré sa majú vykresliť. K jednotlivým objektom má asociované materiály. S prvkami scény manipuluje `BoardController`.
- **`BoardController`** – Spracováva udalosti vykonané hráčom. Tieto udalosti spracuje, a transformuje ich na interné udalosti. Ak sa jedná o posun, otáčanie alebo priblíženie, tak spolupracuje s `BoardRenderer`. Naopak, ak došlo k výberu políčka alebo voľného miesta, tak komunikuje s rozhraním `GameLogic`.

6.3 Definícia hernej dosky

Framework pri spustení spracuje XML konfiguračný súbor, ktorý definuje hernú logiku, a podľa neho odošle získané informácie, cez JNI, do natívnej časti systému, kde sa inicializuje herný engine a spustí sa hra. Je dôležité, aby naša herná doska uchovala svoj stav, keď dôjde k vypnutiu obrazovky alebo pri prichádzajúcom hovore na mobilnom zariadení. Aktivity resp. Fragmenty na platforme Android sa môžu nachádzať jednom zo štyroch stavov [12]:

- *Aktívna* – aktivita je spustená a beží na popredí, je možné s ňou pracovať.
- *Pozastavená* – Aktivita je spustená, beží, ale jej časť je niečím prekrytá
- *Zastavená* –Aktivita bola spustená, ale teraz je prekrytá inou aktivitou.
- *Mŕtva* – Užívateľ buď aktivitu vôbec nespustil alebo bola ukončená.

Systém Android pri prechádzaní medzi týmito stavmi volá metódy aktivity. V našom frameworku sme počet stavov, na ktoré bude upozornená herná logika, zredukovali na dva – aktívna a neaktívna. Pri prechode medzi stavmi je upozornená herná logika volaním funkcií `onStart` a `onStop`. Herná logika by v rámci týchto funkcii mala uložiť alebo obnoviť svoj stav.

6.3.1 Ukladanie dát v hernej logike

Ak by herná logika nemala možnosť ukladať svoje vlastné údaje, bola by veľmi ochudobnená. API pre ukladanie dát by malo byť čo najjednoduchšie, a malo by umožňovať bežné dátové typy ako sú čísla, pravdivostné hodnoty a reťazce. Platforma Android poskytuje niekoľko spôsobov ako aplikácia môže uložiť svoje dáta.

SQLite databáza

V systéme Android je databázový systém vstavaný do behového prostredia, takže ho môže použiť každá aplikácia. SQLite má malú pamäťovú náročnosť a je relatívne rýchla [15]. Použitie SQLite prináša všetky výhody, ktoré majú relačné databázové systémy pri ukladaní štruktúrovaných dát.

Externé úložisko – zápis do súborov

Niekedy je dostatočné ukladať dáta jednoduchším spôsobom ako je vstavaný databázový systém. Čítanie a zápis do súboru prebieha takmer rovnako, ako na stolnom počítači.

Ukladanie typu kľuč hodnota

Najjednoduchší spôsobom ukladania dát je systém nastavení. Android dovoľuje aktivitám a aplikáciám ukladanie dát štýlom kľúč – hodnota. Kľúč musí byť typu String a hodnoty môžu byť primitívne dátové typy. Uložené nastavenia sa môžu týkať iba jednej aktivity alebo môžu byť zdieľané naprieč celou aplikáciou.

6.3.2 Návrh konfigurácie a využitie XML

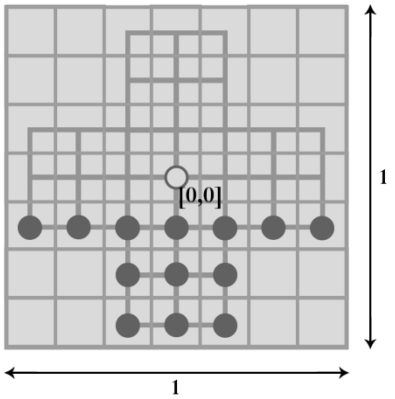
Všetky zásuvné moduly budú konfigurovateľné pomocou XML súborov. Každý XML dokument pozostáva z elementov, ktoré sa v dokumente vyznačujú pomocou tagov – počiatočného a koncového (je voliteľný). Elementy môžu obsahovať text alebo ďalší zanorený element. Týmto spôsobom vzniká hierarchická stromová štruktúra. Okrem elementov je možné na zachytenie informácií použiť atribúty elementov. XML umožňuje používať ľubovoľne pomenované elementy, ale v skutočnosti je lepšie, ak má dokument dopredu predpísanú štruktúru. Štruktúru je možné predpísať pomocou tzv. schém. Medzi najrozšírenejšie jazyky na tvorbu schém patrí *DTD, W3C XML Schéma, RELAX NG* alebo *Schematron* [16]. Hlavný význam XML schém je formálna definícia dátových formátov založených na XML. V našom projekte použijeme W3C XML Schému (ďalej len XML Schéma). Tento formát popisu využíva syntax XML a používa špeciálne elementy patriace do ich menného priestoru[4][5]. Na vytvorenie takejto schémy je možné použiť vizuálne editory, ktoré sú obsiahnuté napríklad v integrovaných prostrediach, ako je Eclipse. Ak chceme aby náš dokument obsahoval informáciu o použitej schéme, tak použijeme globálne atribúty `schemaLocation` alebo `noNamespaceSchemaLocation` z ich menného priestoru[6]. V prípade, že sa nepoužívajú menné priestory, tak stačí do atribútu `noNamespaceSchemaLocation` zapísať cestu k XML schéme alebo URL na ktorej sa nachádza.

[4] Menný priestor: `http://www.w3.org/2001/XMLSchema`
[5] Pre viac informácii o použití navštívte: http://www.w3schools.com/schema/
[6] Menný priestor: `http://www.w3.org/2001/XMLSchema-instance`

Návrh konfigurácie hernej dosky

Teraz bude nasledovať návrh toho, čo sa bude dať definovať, prvky budú reprezentované pomocou XML elementov a atribútov. Pri navrhovaní štruktúry sme sa riadili niekoľkými odporúčaniami pre tvorbu XML schémy. Často sa rieši, či je lepšie na použiť na zaznamenanie informácie element alebo atribút. Zatiaľ čo element je možné ďalej členiť zanorovaním ďalších prvkov, o atribútov to neplatí. Preto, v tomto prípade sme sa držali pravidla, že do atribútov vkladáme hodnoty iba vtedy, ak dopredu vieme, že obor hodnôt je obmedzený a nebude sa meniť [16].

- **Herná logika** – Cesta k hernej logike.
- **Herná doska**
    - **Rozmer hernej dosky** – Nutný atribút udávajúci počet stĺpcov a riadkov.
    - **Geometria hernej dosky** – Názov súboru, ktorý obsahuje geometriu hernej dosky. Mala by obsahovať hernú plochu, prípadne okolie. Bude uložená vo formáte OBJ, ktorý bude pozostávať z vrcholov a indexov. Vrcholy v tomto súbore budú obsahovať pozíciu, normálu, textúrovaciu súradnicu. Stred dosky by mal byť zameraný sa stred súradnicového systému objektu (Obrázok 17). Výška hernej plochy, by mala byť na úrovni 0 (súradnica y). Rozmer menšej strany hernej plochy, by mal byť 1 jednotka.
    - **Material** – názov materiálu, ktorý sa použije.
    - **Zrkadlenie** – Povolenie resp. zakázanie odrazu figúrok od dosky.
    - **Výber políčka –** Nastaví sa, ako má byt vykreslená informácia o tom, že je vybraté políčko. Jednou z možností je prekrytie farbou.



Obrázok 17 Herná doska zasadená do prostredia. Figúrky sa nachádzajú na virtuálnych pozíciách (modré štvorce).

- **Nastavenia** – Bude obsahovať hlavne grafické nastavenia celoplošných grafických efektov. Tieto grafické efekty môžu mať vlastné špecifické atribúty. Napríklad u tieňových máp, by sa mohlo zapínať a vypínať PCF.
- **Figúrky** – Obsahuje zoznam typov figúrok, ktoré sa vyskytujú v scéne. U každého typu bude možné nastaviť niekoľko atribútov.

- **Názov figúrky** – S konkrétnym typom figúrky, sa v kóde pracuje cez jej názov.
- **Geometria figúrky** – podobne ako u geometrie hernej dosky.
- **Definícia stavov figúrky** – figúrka môže byť v niekoľkých stavoch – *normálna, označená* a *nedostupná*. V jednom okamihu je figúrka práve v jednom stave. Povinné je iba definovanie normálneho stavu. Pre každý tento stav, sa nastaví nasledujúci atribút:
  - **Material** – názov materiálu, ktorý sa použije.

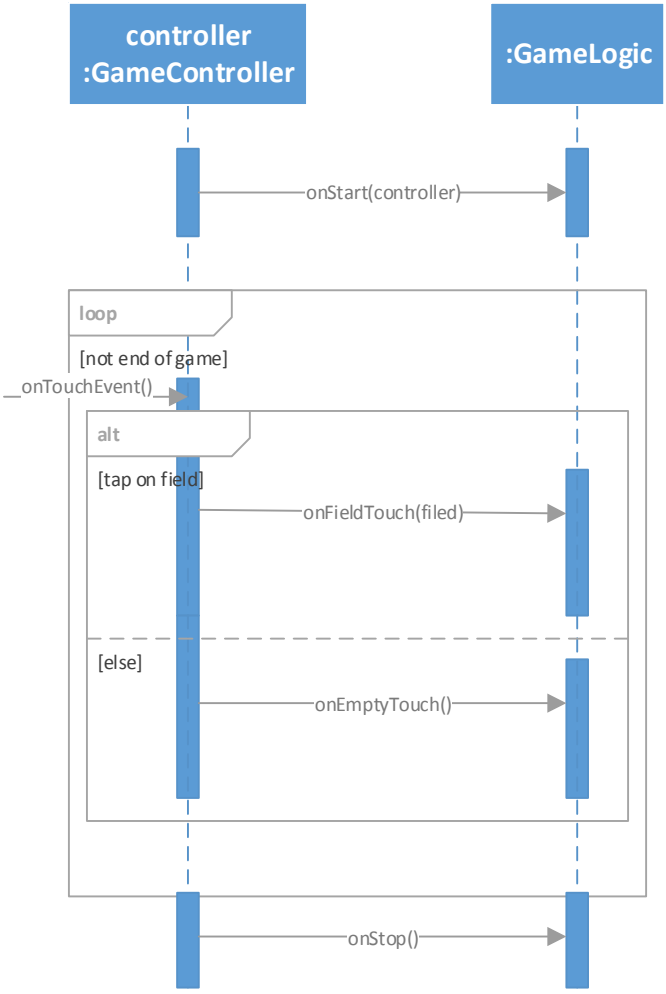
- **Materiály** – Bude obsahovať výpis inštancii materiálov, ktoré sa budú dať použiť na figúrky alebo hracie pole. Inštancie materiálov budú vychádzať z pevne daných základných materiálov. Napríklad, základný material textúrovaného objektu, bude požadovať, aby každá inštancia zadala názov súboru textúry. Iným materiálom bude kreslená textúra, a tá zas bude vyžadovať základnú farbu.
  - **Názov inštancie materiálu** – každý material bude musieť byť pomenovaný týmto atribútom.

6.4 Rozhranie herná logika – BoardEngine

Komunikácia so zásuvným modulom začína zavolaním funkcie `onStart()`. V tejto metóde, by mal plugin inicializovať hernú dosku, prípadne obnoviť stav svoj stav, ak bola herná logika pozastavená. Ako parameter je predaný pointer na objekt `BoardHandler` (Obrázok 18).

Po inicializácii sa čaká na interakciu hráča. Po výbere hracieho poľa alebo voľného miesta sa zavolá nad pluginom funkcia `onFieldTouch()` s parametrom, ktorý obsahuje pointer na vybrané pole, resp. funkcia `onEmptyTouch()` bez parametrov. V rámci týchto volaní, plugin môže pracovať nad hernou doskou skrz tieto metódy:

- `getField(row, column)` – vráti políčko v danom riadku a stĺpci.
- `moveFigureToField(figure, field)` – posunie figúrku na zvolené pole. Príklad použitia možno vidieť v Príloha .
- `createNewFigure(typ)` – vytvorí sa nóva figúrka, ktorá ešte niekde nestojí. Jej typ je špecifikovaný reťazcom a odpovedá typu v konfiguračnom súbore.
- `setFieldState(field, state)` – nastaví políčku jeden z dvoch stavov.
- `setFigureState(figure, state)` - nastaví políčku jeden z troch stavov.
- `moveFigureToAbsolutePosition(figure, row, column)` – figúrka sa presunie na zvolené absolútne miesto. Súčasne sa prepne do stavu *nedostupná*. Figúrka sa už nedá viac použiť.
- `destroyFigure(figure)` – figúrka sa vymaže.



Obrázok 18 Systémový diagram sekvencie. Zobrazuje návrh inicializácie hernej logiky, herný cyklus a pozastavenie hernej logiky.

6.5 Podpora pre vlastnú hernú logiku

Vlastná herná logika, teda modul `GameLogic`, môže byť vytvorená rôzne. V aktuálnej sekcii si zhrnieme rôzne prístupy, ktoré sa líšia použitým programovacím jazykom, dostupným rozhraním alebo prenositeľnosťou.

6.5.1 Java

Prvou, a prirodzenou, možnosťou je implementácia v jazyku Java. Herná logika by musela implementovať rozhranie, používala by poskytnuté objekty prípadne aj Android kontext. Výhodou tohto riešenia by bolo, že Java je hlavný jazyk platformy Android a že klient by mal priamy prístup k API Android. Takto by klient mohol jednoducho implementovať grafické rozhranie pomocou vstavaných prvkov alebo by mohol pracovať so zvukom. Tento prístup poskytuje najväčšie možnosti pri tvorbe hernej logiky. Nevýhodou je, že herná logika by potom musela s herným enginom komunikovať nepriamo cez JNI.

6.5.2 C++

Druhým prístupom by mohlo byľ použitie jazyka C++. Herný modul by bol podobne, ako v predchádzajúcom prípade implementovaný voči rozhraniu. Android API v NDK neposkytuje také možnosti ako jeho Java implementácia. Modul by sa pripájal ako C++ knižnica, volaním Java metódy `Runtime.loadLibrary()`. Výhodou je ľahšie napojenie na herný engine a prenositeľnosť kódu.

6.5.3 Skriptovací jazyk

Poslednou možnosťou, ktorú si predstavíme, je integrácia interprétu skriptovacieho jazyka ako napríklad Lua alebo ECMAScript. Zjavnou výhodou je jednoduchosť ich použitia a teda efektivita vývoja. Nevýhodou je, že do skriptovacieho jazyka by bolo potrebné dodať potrebné API. V pod sekciách si predstavíme niekoľko spôsobov, ako vložiť inteprét skriptovacieho jazyka do Android aplikácie.

WebView

Pohľad `WebView` dovoľuje na Androide zobrazenie webovej stránky. Okrem zobrazovania je schopný interpretovať JavaScriptový kód na stránke, vďaka čomu, by bolo možné vykonávať voliteľný kód, pričom pohľad `WebView` by mohol byť skrytý.

Rhino[7]

Jedná sa o JavaScript implementáciu v jazyku Java, spravovanú organizáciou *Mozilla Foundation*. Použitie je jednoduché, no nevýhodou je nutnosť spúšťať JavaScript kód z Javy.

Duktape[8]

Duktape je interprét jazyka **ECMASript** vo verzií E5/E5.1 a pridáva niekoľko rozšírení. Na rozdiel od predchádzajúcich prístupov sa nejedná o JavaScriptový engine, teda zbytočne neobsahuje doplnky pre prácu s webovými stránkami a má nízke nároku na systém. Jeho integrácia je jednoduchá, pretože sa jedná o knižnicu jazyka C. Na začiatku sa pridajú súbory `duktape.c` a `duktape.h`. Potom sa pri inicializácii programu vytvorí Duktape kontext a načítajú sa skripty. Nakoniec sa na vhodné miesta v programe vložia volania skriptov. Volania prebiehajú vkladaním hodnôt na hodnotový zásobník Duktape kontextu. Po vykonaní skriptu je tok programu vrátený do programu a návratová hodnota je uložená na zásobníku. Duktape ďalej dovoľuje, aby ECMASriptový kód volal C kód, čo sa hodí pre výpočetne náročné úlohy alebo na prístup k funkciám programu.

[7] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/Rhino
[8] http://duktape.org/

6.6 Podpora pre tvorbu GUI

Platforma Android poskytuje široké možnosť v oblasti tvorby užívateľských rozhraní. Preto neprekvapí, že jedným z prístupov je použitie práve Android API. V tomto prípade by musela byť herná logika implementovaná v Jave, a potom užívateľské rozhranie by sa vykresľovalo nad 3D scénu. Spracovanie udalostí a komunikácia s hernou logikou by bola v rukách klienta.

Alternatívou je vložiť podporu pre tvorbu GUI priamo do herného enginu. V kombinácii s použitím skriptovacieho jazyka, by bola potom herná logika úplne oddelená od platformy Android. Podpora pre GUI by mohla byť, v počiatku, obmedzená na zobrazenie obrázkov a naviazaní funkcií na dotyk, na tieto obrázky. Zobrazený obrázok by bola istá forma tlačidla alebo by len zobrazoval stavovú informáciu. Bolo by nutné vyriešiť spôsob umiestnenia obrázku na obrazovku, zachytávanie udalostí dotyku, navrhnúť spôsob deklarácie obrázku a API pre manipuláciu s obrázkom.

7 Implementácia

Aktuálna kapitola pojednáva o popise implementovaných komponent. Zmieni sa o dôvode ich vzniku, implementácii a o spôsobe ich použitia. Ďalej budú diskutované architektonické rozhodnutia prijaté pri vývoji. Kapitola bude rozčlenená na opis podpornej knižnice HOGL, určenej na zjednodušenie práce s knižnicou OpenGL na platforme Android (7.3), potom sa zameriame na samotný grafický engine (7.4) a nakoniec sa ponoríme do vysvetlenia implementovaného zásuvného modulu – hry Líška a husi, ktorá anglickou variáciou na hru Vlk a ovce (7.5).

Práca začala na základe už existujúcej knižnice autora HOGL, ktorá bola značne rozšírená počas implementácie technickej práce (menovite sú to C++ moduly `extensionTools`, `fileTools`, `frameBuffer`, `objParser`, `textureTools` a `vertexBuffer`). Vývoj grafického enginu prebiehal hlavne na platforme Windows a implementovaná funkcionalita sa priebežne testovala na zariadeniach a emulátore s OS Android. Aby sme boli schopný volať OpenGL ES 2.0 príkazy na platforme Windows, tak sme použili knižnicu ANGLE. Knižnica je implementácia OpenGL ES 2.0 špecifikácie, ktorá OpenGL volania prekladá na DirectX 9 alebo DirectX 11 API volania [17]. Ako prvý bol vyvinutý grafický engine, nasledovalo vykresľovanie hernej plochy, potom boli do grafického enginu pridané grafické efekty a nakoniec vznikol zásuvný modul.

7.1 Architektúra systému

Na najvyššej úrovni sa systém skladá z časti implementovanej v manažovanom kóde a časti vytvorenej v jazyku C++. Medzi oboma časťami prebieha komunikácia cez JNI. Manažovaný kód sa stará hlavne o napojenie na platformu Android pomocou triedy `BoardFragment`. Ďalej obsahuje pohľad, ktorý zobrazuje 3D scénu – `BoardView`. `BoardView` je potomkom triedy `GLSurfaceView` a tak sprístupňuje OpenGL kontext aplikácii. Okrem iného zaisťuje detekciu gest a vlastní objekt typu `BoardRenderer`. `BoardRenderer` má za úlohu vykresliť 3D scénu, no implementácia vykresľovania je v už C++ kóde. Okrem toho predstavuje rozhranie do natívnej strany kódu (obsahuje deklarácie natívnych funkcií). Manažovaný kód ešte sprístupňuje podporu pre ukladanie údajov a parsovanie konfiguračného súboru, čím vzniká objekt typu `BoardConfig`, ktorý obsahuje konfiguráciu zásuvného modulu.

Natívna strana frameworku obsahuje väčšinu funkcií. Centrom je modul `engine.cpp`. Obsahuje dva menné priestory `renderer` a `controller`, ktoré obsahujú sadu funkcií umožňujúce ovládanie herného enginu. Menný priestor `renderer` obsahuje funkcie týkajúce sa samotného vykresľovania, volané z Java objektu `BoardRenderer` (`onInit`, `onResize`, `onDraw`,

`onExit`). Na druhú stranu menný priestor `controller` slúži na manipuláciu s hernou logikou a nastavenie konfigurácie hernej dosky.

Pre účel komunikácie s hernou logikou vznikla trieda `GameLogicWrapper` a jej úlohou je zapuzdrenie interpétu jazyka ECMAScript, načítanie skriptov a preposielanie príkazov do hernej logiky.

7.1.1 Vlákna

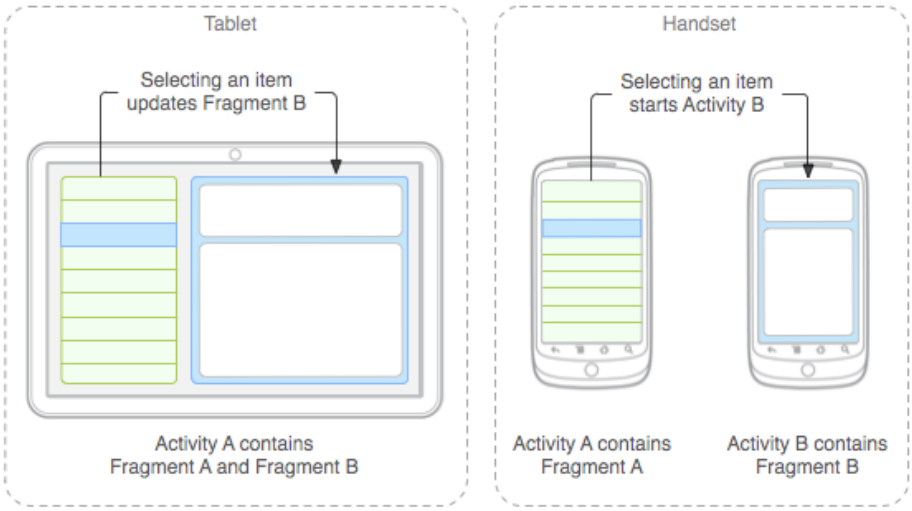
Náš framework používa má spustené dva základné vlákna. Prvým je vlákno užívateľského rozhrania, ktoré napríklad deleguje dotykové udalosti. Druhým vláknom je vlákno OpenGL kontextu, ktoré vzniká pri použití pohľadu `GLSurfaceView` a jeho úlohou je v cykle vykresľovať snímky pomocou OpenGL knižnice. Je to jediné vlákno, ktoré môže pracovať s OpenGL kontextom. Okrem spomínaných vlákien existuje vlákno časovača pre hernú logiku.

7.2 Zásuvný modul

Android aplikácia, ktorá používa náš framework, typicky implementuje jeden zásuvný modul, aby mohla byť zobrazená daná herná doska pre konkrétnu hru. Tento zásuvný modul je integrovaný do aplikácie vo forme Android komponenty, zvanej **Fragment**. Na platforme Android existujú dve základné komponenty užívateľského rozhrania – Aktivita a Fragment. **Aktivita** je dôležitou súčasťou životného cyklu aplikácie. Reprezentuje jednu obrazovku užívateľského rozhrania. Aktivity spolu tvoria rozhranie aplikácie, ale súčasne sú na sebe nezávislé [18]. Fragment reprezentuje chovanie alebo časť užívateľského rozhrania Aktivity. V jednej Aktivite môže byť skombinovaných niekoľko Fragmentov, aby bolo možné vytvoriť viac panelovú obrazovku a zároveň, jeden Fragment sa dá použiť vo viacerých Aktivitách. Fragment musí byť vždy vložený v Aktivite a jeho životný cyklus je priamo ovplyvnený životným cyklom Aktivity [19].

Pre použitie zásuvného modulu je potrebné vložiť **`BoardFragment`**, ktorý dedí z triedy `Fragment`, do Aktivity aplikácie. `BoardFragment` okrem svojej základnej úlohy, zobrazenie a práca s hernou doskou, umožňuje preposlanie príkazov do hernej logiky skrz metódu `executeCommand`, ktorá ako parameter prijíma reťazec s JavaScript kódom. Týmto sa uľahčuje prepojenie aplikácie so zásuvným modulom a súčasne poskytuje spôsob ladenia hernej logiky pri vývoji. Keďže Fragment je oboznámený so zmenami životného cyklu Aktivity, tak na tieto zmeny je upozornený aj zásuvný modul. Zvolený prístup, keď plugin je reprezentovaný Fragmentom, dovoľuje vybudovanie komplexnej aplikácie, napríklad, použitie prvkov užívateľského rozhrania z Androidu na ovládanie častí hernej logiky (i keď jednoduché ovládacie prvky sú zabudované priamo do herného enginu), prípadne napojenie na iné Android API.

V rámci aplikácie je možné použiť niekoľko zásuvných modulov – BoardFragmentov. Každý z nich je nakonfigurovaný pomocou vlastného konfiguračného XML súboru (viac neskôr). Názov konfiguračného súboru sa vkladá cez konštruktor triedy `BoardFragment` a mal by byť umiestnený v priečinku `/assets`.



Obrázok 19 Príklad, ako je možné vytvoriť modulárne užívateľské rozhranie pomocou Fragmentov[9]. Vľavo je kombinácia dvoch Fragmentov v jednej Aktivite na obrazovke tabletu. Vpravo sú oba fragmenty vložené do samostatných Aktivít.

7.2.1 Použitie enginu

Použitie enginu v aplikácii je priamočiare, a pozostáva z niekoľkých krokov:

1. Náš engine je Eclipse projekt, predstavujúci Android knižnicu. Pripojenie enginu do projektu aplikácie potom znamená, pripojenie tejto knižnice ako projektovej závislosti.
2. Po pripojení je dostupná trieda `BoardFragment` v balíčku `sk.kasper.boardengine`, ktorá sa použije v aplikácii bežným spôsobom (napríklad sa vloží do počiatočnej Aktivity aplikácie). Aktivita, ktorá obsahuje Fragment hernej dosky by mala mať napevno nastavenú orientáciu obrazovky.
3. Vytvoria sa súbory, ktoré sú potrebné v zásuvnom module. Jedná sa o obrázky na textúry a tlačidlá, geometriu pre hernú dosku a figúrky. Tieto súbory sa vkladajú do priečinka `/assets`.
4. Naprogramuje sa herná logika v jazyku ECMAScript, ktorá implementuje požadované funkcie a jej zdrojové súbory sa vložia do priečinka `/assets`.
5. Nakoniec sa napíše konfiguračný súbor, ktorý využije cesty k súborom vytvorených v krokoch 3 a 4. Tiež sa vloží do priečinka `/assets`. Názov konfiguračného súboru sa predá továrňovej metóde `newInstance` triedy `BoardFragment`.

[9] Zdroj: http://developer.android.com/guide/components/fragments.html

7.2.2 Integrácia jazyka ECMAScript

Aby sme v maximálnej miere uľahčili tvorbu zásuvného modulu, integrovali sme do nášho enginu interpret jazyka ECMAScript. Voľba padla na engine **Duktape** hlavne z dôvodu prenositeľnosti (vývoj na platforme Windows), jednoduchosti integrácie, nízkej náročnosti na zdroje a malej veľkosti.

Celý interpret je napísaný v jazyku C a pozostáva len z jedného hlavičkového a jedného zdrojového súboru. V nasledujúcom popise si predstavíme časti interpretu významné pre naše účely. Pri vývoji, vo väčšine prípadov, je vhodné, ak je zdrojový kód rozdelený do viacerých súborov. Duktape umožňuje vytváranie a načítanie modulov rovnakým spôsobom ako *CommonJS*[10]. Modul sa načítava volaním `require`, ktorý vráti objekt modulu, cez ktorý môžeme pristupovať k exportovaným položkám, ako možno vidieť na ukážke nižšie. Načítanie súborov je samozrejme platformovo závislé, preto podporu pre načítanie súborov je nutné implementovať samostatne. Náš framework očakáva, že moduly sa nachádzajú v adresáre `/assets`, podobne ako hlavný súbor hernej logiky.

```
// module loading
var mod = require('foo/bar');
mod.hello();
```

```
// foo/bar.js
// not visible outside the module
var text = 'Hello world!';

// loads foo/quux
var quux = require('./quux');

exports.hello = function () {
    print(text);
};
```

Zdroj. kód 1 Prestavenie práce s modulmi[11]. Vľavo je ukázané načítanie súboru `foo/bar.js`. Vpravo je definícia tohto modulu. Z vonka sú viditeľne iba položky priradené do objektu exports.

Interpret obsahuje funkcie, ktorými sa dajú objekty serializovať a deserializovať vo formáte JSON alebo vo vlastných JSON formátoch JX a JC[12]. Prevod do týchto formátov má hlavne význam pri ukladaní stavu hernej logiky.

7.2.3 Životný cyklus hernej logiky

Ako bolo spomenuté, stav hernej logiky, z pohľadu herného enginu, sa dá vidieť v dvoch stavoch: aktívny a zastavený. Pri prechode medzi stavmi sú volané funkcie `onStart` alebo `onStop`, ktoré sú implementované hernou logikou. Nie je nutné ich programovať, ale je to vhodné

[10] V klientskom JavaScrit kóde sa moduly načítavajú pomocou HTML tagu `<script>`. Pri použití JavaScriptu na servery, takáto možnosť neexistuje. *CommonJS* obsahuje preto spôsob, ako načítavať moduly cez volanie funkcie `require`.
[11] Ukážka prevzatá z http://duktape.org/guide.html.
[12] http://duktape.org/guide.html#customjson

miesto na uloženie alebo obnovenie interného stavu hernej logiky. Herná logika môže byť voliteľne komplexná (rôzne obrazovky, stavy implementovanej hry atď.), a preto je vhodné uchovať vnútorný stav pre prípad straty Aktivity aplikácie.

Keď je herná logika v aktívnom stave, tak prijíma udalosti o výberu políčka a dotyku na prázdne miesto a obrazovke. Reakcia na spomínané udalosti by mali byť vložené do funkcií `onFieldTouch` a `onEmptyTouch`. Tieto funkcie teda slúžia ako hlavné vstupné body pre implementáciu hernej logiky. Inou formou, ako získať odozvu od hráča, je vstavaná podpora pre GUI (viac v sekcii Podporné API pre hernú logiku).

7.2.4 Konfigurácia hernej dosky

Položky, ktoré je možné definovať v konfiguračnom súbore boli popísané v sekcii 6.3. Konfigurácia zásuvného modulu bude písaná v značkovacom jazyku XML. Značky a pravidlá, ktoré je nutné dodržiavať v rámci konfiguračného súboru sú definované v XML schéme, ktorá bola pre tento účel vytvorená. Táto schéma sa nachádza v súbore `board.xsd`. Vo fragmente kódu nižšie možno vidieť minimálny konfiguračný súbor zásuvného modulu. Komplexnejší príklad konfigurácie je možné zahliadnuť v zdrojovom kóde ku hre *Fox & Geese*.

```
<?xml version="1.0" encoding="UTF-8"?>
<board xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="board.xsd">
    <gameLogic>logic.js</gameLogic>
    <materialName>board_material</materialName>
    <layout>
        <squared>
            <width>7</width>
            <depth>7</depth>
        </squared>
    </layout>
    <geometry>
       <source path="board_fox_and_geese.obj"/>
    </geometry>
    <materials>
         <solidMaterial name="board_material">
              <glossines>0.2</glossines>
               <image path="board_fox_and_geese.png"/>
          </solidMaterial>
    </materials>
</board>
```

Zdroj. kód 2 Ukážka minimálnej konfigurácie zásuvného modulu. Definícia značiek v defaultnom mennom priestore pochádza zo schémy v súbore `board.xsd`. V kóde je možné postupne vidieť zadaný súbor hernej logiky, material hernej dosky, počet políčok hernej dosky, súbor s geometriou hernej dosky a definíciu materialu hernej dosky.

Konfigurácia, tak ako je popísaná v schéme, požaduje koreňový element *board*, ktorý na obsahuje, na najvyššej úrovni, elementy *gameLogic*, *layout*, *materialName*, *geometry*, *fieldSelection*, *graphicEffects*, *figures*, *imageRectangles*, *materials* a *background*. Význam všetkých elementov je popísaný v tabuľke nižšie.

Názov	Povinný	Význam	Podelementy
gameLogic	Áno	Názov súboru s hernou logikou.	-
layout	Áno	Spôsob rozloženia políčok.	squad
squad	Nie	Šachovnicový typ dosky.	width, depth
materialName	Áno	Názov materiálu, ktorý sa aplikuje na hernú dosku	-
geometry	Áno	Definícia geometrie hernej dosky	source
source	Nie	Cesta k zdrojovému súboru geometrie hernej dosky vo formáte OBJ. Názvov súboru sa vkladá do parametru *path*.	-
fieldSelection	Nie	Spôsob zvýraznenia vybraného políčka.	colorOverlay
colorOverlay	Nie	Vybrané políčko bude prekryté farbou.	color
color	Nie	Definícia farby. Vnorené elementy určujú jednotlivé RGBA zložky	red, green, blue, alpha
graphicEffects	Nie	Určuje vlastnosti grafických efektov. Každý z vnorených elementov obsahuje atribút *enabled* pre zapnutie, resp. vypnutie efektu.	depthOfField, shadowMap
depthOfField	Nie	Umožňuje zmeniť intenzitu efektu hĺbky ostrosti, ovplyvňuje aj bloom efekt.	strength
strength	Nie	Element určujúci intenzitu rodičovského elementu. Hodnoty sú v rozsahu <0, 1>	
shadowMap	Nie	Nastavenie efektu tieňa.	PCF
PCF	Nie	Dovoľuje zapnúť/vypnúť vyhladzovanie tieňov cez atribút *enabled*.	
figures	Nie	Definícia figúrok, ktoré je možné vkladať na hernú dosku.	figure
figure	Nie	Definícia jednej figúrky	geometry, states
states	Áno	Popis stavov, v ktorých sa môže figúrka nachádzať, musí obsahovať aspoň jeden stav.	state
state	Áno	Je odkazovateľný pomocou mena zadaného cez atribút *name*. Je charakterizovaný použitým materiálom.	materialName
imageRectangles	Nie	Definuje zoznam obrázkov, ktoré sa môžu používať v GUI napr. tlačidlá.	imageRectangle
imageRectangle	Nie	Popisuje jeden obrázok, na ktorý sa odkazuje cez meno zadané v atribúte *name*.	image, zIndex
image	Áno	Určuje obrázok, ktorý sa použije v rodičovskom elemente. Názov obrázku sa zadáva cez atribút *path*. Je možné použiť obrázky v komprimovanom formáte ETC1.	-
zIndex	Nie	Určuje poradie vykresľovania prekrývajúcich sa obrázkov. Čim vyššie číslo, tým vyššie je obrázok. Základná hodnota je 0.	-
materials	Áno	Definícia typov materiálov. Musí byť uvedený aspoň jeden. Na materiály sa odkazuje cez ich meno uvedené v atribúte *name*.	solidMaterial, toonMaterial

Názov	Povinný	Význam	Podelementy
solidMaterial	Nie	Základný material, používajúci Phongov osvetľovací model. Farba je získaná z textúry.	glossines, image
glossines	Nie	Určuje úroveň lesklosti materiálu.	
toonMaterial	Nie	Material vykresľujúci kreslený objekt. Základná farba pochádza z obrázku.	image, toonCount
toonCount	Nie	Určuje počet odtieňov základnej farby.	
background	Nie	Definuje pozadie scény.	color

Tabuľka 2 Popis elementov, ktoré je možné použiť v konfiguračnom XML súbore.

Spracovanie údajov z konfiguračného súboru sa vykonáva pomocou Java knižnice **Simple**[13]. Knižnica Simple XML je serializačný a konfiguračný XML framework a umožňuje rýchly vývoj XML konfigurácie s minimálnou námahou a redukuje množstvo chýb, pretože je založený na anotáciách (viď príklad nižšie), ktorými sú popísané dátové entity a ich vlastnosti [20]. Po prečítaní, je konfigurácia odoslaná cez rozhranie JNI do natívnej časti aplikácie, kde je podľa nej nastavený počiatočný stav enginu.

```
@Root
public class Layout {
   @Element
   private String width;

   @Element
   private String depth;
}
```

```
<layout>
    <width>7</width>
    <depth>8</depth>
</layout>
```

Zdroj. kód 3 Vľavo ukážka popisu jednoduchej dátovej entity pomocou Java anotácii[14] vo frameworku Simple. Napravo je odpovedajúca časť XML dokumentu.

7.2.5 Ukladanie stavu hernej dosky

Pre uľahčenie tvorby hernej logiky, je pridaná podpora pre automatické ukladanie a obnovu stavu hernej dosky. Je nezávislá na ukladaní interných hodnôt v hernej logike. Stav hernej dosky pozostáva z figúrok a ich stavov. Do úložiska sa vždy uloží ako kľúč zakódovaná pozícia figúrky a ako hodnota dvojica typ figúrky a jej stav. Okrem toho sa ukladá aj príznak o uložení. Pri obnovovaní hernej plochy sa najprv podľa príznaku o uložení zistí, či bola hracia plocha uložená, a ak áno, tak sa odstráni príznak a načítajú sa všetky figúrky z úložiska. Ukladanie prebieha po volaní ECMAScriptovej funkcie `onStop` a obnovenie pred funkciou `onStart`.

7.2.6 Podporné API pre hernú logiku

Aby bola tvorba na hernej logiky možná s využitím jazyka ECMAScript, tak je nutné poskytnúť klientovi programové rozhranie umožňujúce jeho tvorbu. Do hernej logiky boli vložené definície tried a funkcii, menovite: API pre manipuláciu s hernou doskou, časovač, prácu s GUI

[13] http://simple.sourceforge.net/
[14] Zdroj: http://simple.sourceforge.net/download/stream/doc/examples/examples.php

a ukladanie hodnôt kľúč – hodnota. Rozhranie je implementované ako obálka nad C++ kódom s využitím podpory knižnice Duktape. V krátkosti si predstavíme možnosti a spôsob jeho použitia.

Manipulácia s hernou doskou

API pre manipuláciu s hernou doskou sa nachádza v objekte `board`. V prostredí grafického enginu je postačujúce, že práca s figúrkami je riešená procedurálnym prístupom. Do hernej logiky bol však použitý čiastočne objektovo orientovaný prístup a s figúrkou sa manipuluje ako s objektom. Pri vytvorení objektu sa mu do konštruktoru predá názov typu a názov stavu. Takto vzniknutý objekt sa predáva do funkcie: `placeOnField`, ktorá položí figúrku na zvolené políčko alebo funkcie `takeFromField`, ktorá odoberie zo zadaného políčka figúrku. Obe funkcie vracajú objekt figúrky, ak zvolené políčko nebolo prázdne. Ďalšie funkcie z objektu `board` umožňujú označovanie herného políčka cez funkciu `setFieldSelected` a od označovanie cez `deselectField`. Názorná ukážka použitia je vo fragmente kódu nižšie.

```
var oldX = 0, oldY = 0;

function onFieldTouch(x, y) {
    // posun figurky [oldX, oldY] -> [x, y]
    var f = board.takeFromField(oldX, oldY);
    board.placeOnField(f, x, y);
    board.setFieldSelected(x, y);
    f.setState("selected");

    oldX = x; oldY = y;

    new Timer(1000, function() { // zavola sa po 1 sekunde
        board.deselectField(); // odoznaci sa policko
        f.setState("normal"); // vrati sa figurka do normalneho stavu
    });
};
```

Zdroj. kód 4 Ukážka posunu figúrky po hernej doske a ukážka použitia časovača.

Časovač

Užitočnou funkciou je spustenie zadanej udalosti po uplynutí časového intervalu. Takto je možné riešiť rôzne oneskorenia alebo aj animácie grafického užívateľského rozhrania. Pre účely hernej logiky bol implementovaný časovať. Časovač je implementovaný ako C++ multimapa, kde kľúčom je časový bod a hodnotou je číslo udalosti. Na prácu s hodinami, časovými bodmi a časovými úsekmi bola použitá funkcionalita z menného priestoru `std::chrono`. Udalosti spúšťa vlákno časovača. Vlákno je uspaté na do doby, kým nenastane prvá udalosť alebo kým nie je pridaná nová udalosť cez metódu `addEvent`. Komunikácia s vláknom prebieha pomocou synchronizačného primitíva – podmienenej premennej. Vznik novej udalosti sa načasuje vytvorením objektu `Timer` v hernej logike, ktorému je predaný čas v milisekundách a callback funkcia, ktorá je zavolaná po vypršaní časovača (Zdroj. kód 4). Každá udalosť má interne priradené unikátne ID, ktoré tvorí väzbu medzi internou reprezentáciou v C++ kóde a objektom typu `Timer` v JavaScript kóde.

Manipulácia s GUI

Možnosti práce s GUI v hernej logike sú síce obmedzené, ale v mnohých prípadoch postačujúce. Programátor má príležitosť zobraziť na obrazovke bitmapový obrázok. Taký obrázok eventuálne slúži iba ako estetický prvok alebo je možné nastaviť mu chovanie, ktoré sa vyvolá po kliknutí.

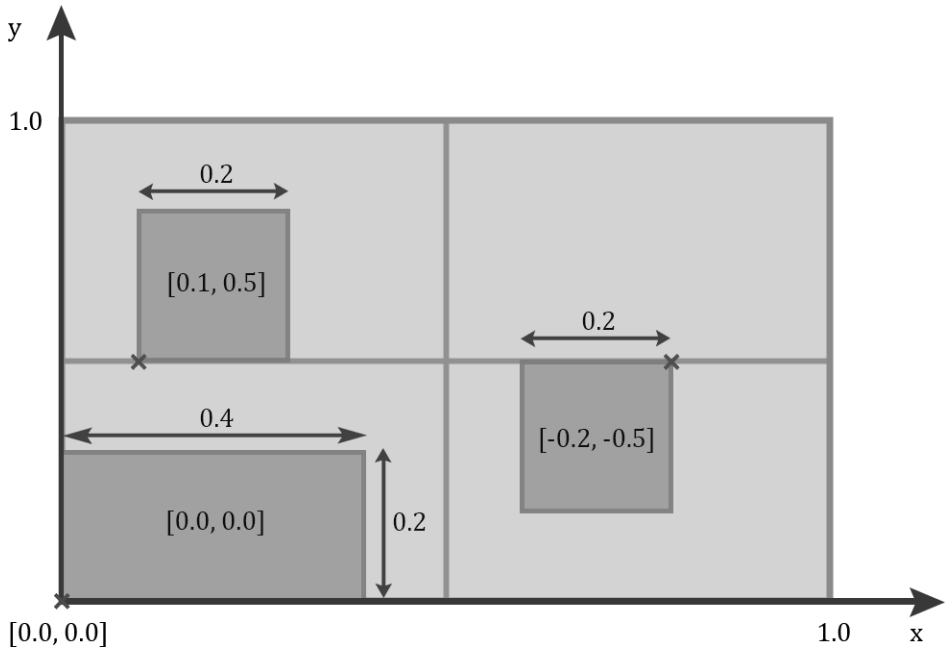
Kód pre prácu s GUI je umiestnený v objekte `gui` a obsahuje definíciu jedného typu – `ImageRectangle`. Objekt typu `ImageRectangle` reprezentuje obrázkovú bitmapu na obrazovke, ktorej je možné nastaviť pozíciu, rozmery a chovanie po kliknutí. Umiestnenie a rozmery obrázku sa zadávajú v relatívnych jednotkách. Počiatok súradnicového systému je v ľavom dolnom rohu a rozmer každej strany obrazovky je 1 jednotka. Užívateľ má voľbu použiť záporné

```
rec1 = new gui.ImageRectangle("image");
rec1.setX(0.1); rec1.setY(0.5);
  rec1.setWidth(0.2);

rec2 = new gui.ImageRectangle("image");
rec2.setX(-0.2); rec2.setY(-0.5);
rec2.setWidth(0.2);

rec3 = new gui.ImageRectangle("image");
rec3.setX(0.0); rec3.setY(0.0);
rec3.setWidth(0.4);
rec3.setHeight(0.2);
```

Zdroj. kód 5 Ukážka kódu, ktorá vykreslí obdĺžniky na obrázku nižšie.



Obrázok 20 Príklad umiestnenia obrázkov na obrazovku. Je možné vidieť relatívne umiestňovanie, použitie záporných hodnôt a dopočítanie výšky obrázku (výška oboch štvorcov je dopočítaná). Červený kríž určuje počiatočný bod obrázku, voči ktorému sa počíta jeho poloha na obrazovke.

súradnice pri určení pozície obrázku. V tomto prípade sa obráti počiatok súradnicového systému. Napríklad pri zadaní zápornej súradnice x sa počíta umiestňuje od pravého okraja obrazovky. Bod

voči ktorému sa počíta pozícia obrázku je v jeho ľavom dolnom rohu. Pri zadaní zápornej pozície bodu sa mení (Obrázok 20 a Zdroj. kód 5). Výška obrázku môže byť rovná nule (implicitná hodnota), vtedy je výška dopočítaná zo šírky tak, aby bol zachovaný pomer strán[15].

Ukladanie typu kľuč – hodnota

Aby bolo možné jednoduchým spôsobom ukladať dáta, bolo v hernej logike vytvorený objekt `storage`. Obsahuje metódu `persist` s parametrami kľuč a hodnota a samozrejme opačnú metódu `get` prijímajúca kľuč hodnoty, ktorú chceme získať a pripadne aj hodnotu, ktorá bude vrátená v prípade, že dátová položka nie je definovaná. Rozhranie je rovnaké pre všetky dátové typy.

```
var fox = {
    position: [0, 6],
    state: "selected"
}

// serializacia
var fox_str = Duktape.enc('jx', fox)
// ulozenie retazca
storage.persist("fox", fox_str)
// ulozenie priznaku
storage.persist("fox_stored", true)
```

```
// defaultna hodnota je false
var fox_stored =
storage.get("fox_stored", false)

var fox = {
    position:[3, 3],
    state:"normal"}

if (fox_stored) {
    var fox_str = storage.get("fox")
    // deserializacia
    fox = Duktape.dec('jx', fox_str)
}
```

Zdroj. kód 6 Ukážka ukladania typu kľúč – hodnota. Vľavo možno vidieť serializáciu objektu a ukladanie objektu. V pravo je fragment kódu, ktorý sa snaží načítať uloženú hodnotu.

Ukladaná hodnota môže byť typu bool, double alebo string. Pri ukladaní samotnej hodnoty je nutné interne uložiť aj jej typ. Pri volaní metódy `get` sa najprv zistí typ uloženej hodnoty, potom sa zavolá príslušná Java pomocná metóda cez JNI (`getNumber`, `getBool`, `getString`). Vítaným rozšírením, je podpora zo strany Duktape, serializovať a deserializovať Javascriptové zložené štruktúry. Vďaka tomu je možné ukladať a obnovovať stav ľubovoľných objektov alebo polí. V ukážke zdrojového kódu vyššie možno vidieť prácu s týmto API.

7.3 Podporná knižnica pre prácu s OpenGL

V rámci vývoja programovej časti diplomovej práce bola použitá a rozšírená autorova C++ knižnica pre prácu s OpenGL. Jej názov je HOGL (Helpers for OpenGL) a pôvodne vznikla za účelom jednoduchého znovu použitia kódu aj na iných projektoch, ktoré sa týkajú vykresľovania pomocou knižnice OpenGL na platforme Android. Časť kódu knižnice, ktorá sa týka práce s vektormi a maticami, bola prevzatá z projektu Angle[16].

Knižnica bola budovaná tak, aby ju bolo možné použiť aj na platforme Windows. Niektoré časti kódu sú platformovo závislé, preto sú tieto časti rozdelené, pomocou direktív preprocesoru C++,

[15] Pomer strán je vždy 1:1, pretože OpenGL ES 2.0 iné veľkosti textúr nedovoľuje. Okrem toho, musia byť rozmery v pixeloch v mocninách čísla 2.
[16] https://code.google.com/p/angleproject/

na kód pre Android a kód pre Windows. Príkladom platformovo závislých komponent je načítanie textových súborov, textúr, uvoľňovanie OpenGL zdrojov alebo logovanie. Vďaka tomu, že je možné použiť knižnicu aj platforme Windows, sa zjednodušil vývoj natívnej časti nášho vykresľovacieho enginu. Knižnica je napísaná objektovo orientovaným prístupom, i keď čiastočne používa procedurálny a funkcionálny prístup. Je postavená na verzii C++11 štandardu[17] jazyka C++ a používa z neho inteligentné ukazateľe, lambda výrazy, automatické odvodenie typu výrazov a iné. Príklad použitia HOGL knižnice je na strane 44. Pri práci s textúrami, sú vyžadované volania pomocných Java statických metód, ktoré sa musia manuálne doplniť do projektu. V prípade použitia na platforme Android, je ju nutné pripojiť ako statickú knižnicu. Preklad prebieha pomocou súboru `Android.mk`, ktorý je umiestnený v koreňovom adresáre knižnice.

7.3.1 Načítanie súborov

Čítanie obsahu textových súborov umožňuje trieda `FileManager` a jej metóda `readFileAsString`. Získavať uložené súbory na platforme Android je možné rôznymi spôsobmi. My sme sa rozhodli, pre ukladanie do špeciálnej Android zložky `/assets`. K týmto súborom sa pristupuje cez existujúcu Java triedu `AssetManager`[18]. Inštancia tejto triedy je predaná vykresľovaciemu enginu pri spustení aplikácie cez JNI a následne vložená do inštancie triedy `FileManager`.

7.3.2 Načítanie textúr

Pre načítanie textúry je zvolený iný prístup, ako pri načítaní textových súborov. Pre tento účel bola vytvorená trieda `TextureFactory`, ktorá tvorí objekty typu `Texture`. Samotné načítanie obrázku a tvorba OpenGL textúry prebieha v pomocnej Java triede `TextureHelper`, ktorej sa cez JNI predá id OpenGL textúry, do ktorej sa má nahrať obrázok. Tento spôsob bol zvolený preto, lebo na Java strane existujú v Android frameworku triedy, ktoré uľahčujú načítanie komprimovaných textúr. Trieda `Texture` je potom jednoduchá obálka pre manipuláciu s OpenGL textúrou. Naprieč aplikáciou sa pracuje práve s jej inštanciami. Práca s ňou sa zaháji volaním metódy `bind`, ktorá naviaže obsiahnutú textúru na OpenGL kontext.

7.3.3 Práca s vykresľovanou geometriou

Základnou jednotkou, ktorá dovoľuje vykresliť geometriu (zoznam vrcholov s údajmi o pozícii, smere normál a o UV koordinátach) je trieda `VertexBuffer`. Jej rozhranie poskytuje metódy pre nastavenie zoznamu vrcholov a zoznamu indexov. Potom, pri vykresľovaní to sú metódy `bind`

[17] http://www.stroustrup.com/C++11FAQ.html
[18] http://developer.android.com/reference/android/content/res/AssetManager.html

(naviaže vertex buffer na OpenGL kontext) a `draw`. V metóde `draw` je volaná funkcia `glDrawElements`, ale predtým sú nastavené dáta vrcholov cez funkciu `glVertexAttribPointer`. V aplikácii sa dodržiava konvencia, že jednotlivé indexy vertex atribútov (normály, pozície a UV koordináty) sú pevne priradené pomocou funkcie `glEnableVertexAttribArray`.

Údaje o vrcholoch sa dajú zadávať ručne alebo cez triedu `ObjParser`. Trieda `ObjParser` na vstupe očakáva reťazec s geometrickými údajmi vo formáte OBJ (očakáva sa istý spôsob uloženia údajov pri exporte, viď návod v prílohe), ktorý následne parsuje a vzniká zoznam vrcholov a indexov. Získaný výstup z tejto triedy sa dá priamo použiť ako vstup pre `VertexBuffer`.

7.3.4 Práca s shadermi

Keďže práca s OpenGL shadermi často býva repetívna a náchylná na vznik chýb, tak z tohto dôvodu, vnikli v module shaderTool triedy `Shader` a `ShaderProgram`. Trieda `Shader` má schopnosť načítať, kompilovať a verifikovať OpenGL shadery. Ako parametre konštruktora očakáva zdrojový kód shaderu a typ shaderu (vertex alebo fragment shader). Trieda `ShaderProgram` dovoľuje pridávanie inštancií triedy `Shader`, linkovanie shaderov a nastavovanie hodnôt uniformných premenných. `ShaderProgram` očakáva presné pomenovanie atribútových premenných. Shader program sa stáva pripraveným na použitie volaním metódy `use`. Hodnoty uniform premenných sa nastavujú skrz metódy `setUniform*`. Trieda si interne uchováva pozície uniformných a atribútových premenných.

7.3.5 Práca s FrameBuffermi

Pre zjednodušenie manipulácie s OpenGL framebuffermi existuje v knižnici HOGL trieda `FrameBuffer`. Jej hlavnou úlohou je priamočiare programovanie grafických algoritmov, ktoré vyžadujú viac priechodov, volaním niekoľkých metód. Je možné použiť inštanciu triedy `FrameBuffer` ako cieľ vykresľovania a následne použiť jeho obsah, ako vstup v ďalšom priechode grafického algoritmu. K framebufferu sa dá pripojiť *color buffer* a *depth buffer* cez metódy `attachColorBufferAsTexture` resp. `attachDepthBuffer`. Po pripojení je možné overiť kompletnosť zostaveného framebuffera. Pre nastavenie daného framebuffera ako cieľa vykresľovania stačí zavolať metódu `bind`. Potom, ak chceme použiť jeho obsah ako textúru pri ďalšom vykresľovaní, stačí zavolať `bindTexture`.

```
unique_ptr<VertexBuffer> board;
shared_ptr<ShaderProgram> shaderProgram;
unique_ptr<Texture> texture;

onInit(...) {
    ...
    // nacitanie geometrie zo suboru do vertex bufferu
    ObjParser boardParser(fileManager->readFileAsString("board.obj"));
    boardGemetry = make_unique<VertexBuffer>();
    boardGemetry->setVertices(
        boardParser.getVerticies(), boardParser.getVerticiesCount());
    boardGemetry->setIndices(
        boardParser.getIndicies(), boardParser.getTrianglesCount();

    // nacitanie shaderov
    auto shaderVertCode = fileManager->readFileAsString("shader.vert");
    auto shaderFragCode = fileManager->readFileAsString("shader.frag");
    unique_ptr<Shader> vertexShader = make_unique<Shader>(
        shaderVertCode, GL_VERTEX_SHADER));
    unique_ptr<Shader> fragmentShader = make_unique<Shader>(
        shaderFragCode,  GL_FRAGMENT_SHADER));

    // vytvorenie shader programu
    shaderProgram = make_shared<ShaderProgram>();
    shaderProgram->attachShader(*vertexShader);
    shaderProgram->attachShader(*fragmentShader);
    shaderProgram->link();
    shaderProgram->use();
    shaderProgram->setUniformMatrix4v("MVP", Matrix4::identity().data);
    shaderProgram->setUniform("text_id", 0);

    // nacitanie textury
    texture = pTexFactory->loadTexture("image.png");
    ...
}

onDraw() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    texture->bind(0);
    shaderProgram->use();

    boardGemetry->bind();
    boardGemetry->draw();
}
```

Zdroj. kód 7 Ukážka použitia HOGL framewoku na vykresľovanie. V ukážke je možné vidieť inicializačnú časť: načítanie geometrie, vytvorenie shader programu a načítanie textúry. A nakoniec , vo funkcii `onDraw` je vykreslenie snímku.

7.4 Grafický engine

Funkcia grafického enginu pozostáva z: kreslenia hernej dosky, herných figúrok; aplikácie grafických efektov; podpory pre výber objektov v scéne a manipulácie s hernou doskou. Väčšina kódu je implementovaná v jazyku C++, ale napríklad odchytávanie udalosti prebieha v manažovanom kóde. V ďalšom výklade si priblížime niektoré komponenty grafického enginu.

7.4.1 Inicializácia a uvoľnenie grafického enginu

Postupnosť krokov pri inicializácii je nasledujúca:

1. Vytvorí sa továrnička na textúry typu `TextureFactory` a pomocný objekt na prácu so súbormi typu `FileManager`.
2. Je zavolaná metóda `onInit`, ktorá vytvorí typy materiálov, podporné shader programy a niektoré prvky geometrie.
3. Sú nastavené parametre hernej dosky získané z konfiguračného súboru: veľkosť hernej dosky, farba pozadia, materiály, typy figúrok, vzhľad hernej dosky, prvky GUI a herná logika
4. Je zavolaná metóda `onResize`, v ktorej sa aktualizuje objekt kamery, transformačné matice kamery, scény a svetla (pre algoritmus tieňových máp). Inicializujú sa fragment buffre. Upraví sa bod zaostrenia pre DOF efekt.

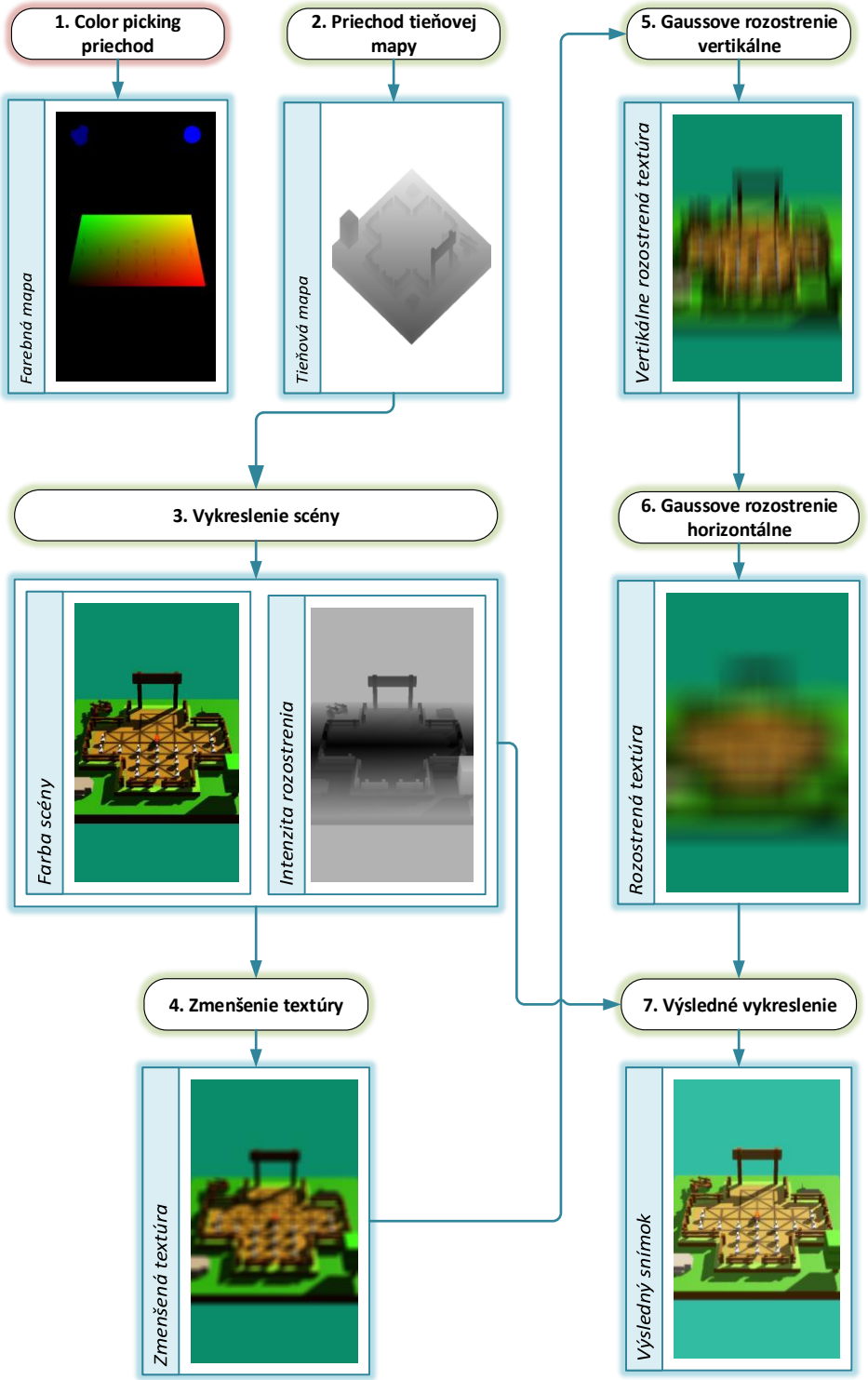
O ukončenie grafického enginu sa stará funkcia `onExit`, a jej úlohou je uvoľniť alokované zdroje. Najprv sa zastaví herná logika a uloží sa stav hernej dosky, potom sa ukončí vlákno časovača a nakoniec sa vyprázdnia kolekcie, ktoré obsahovali figúrky, textúry, materiály atď. OpenGL zdroje sa neuvoľňujú, k ich uvoľneniu dochádza automaticky[19].

7.4.2 Vykreslenie snímku

Vykreslenie scény prebieha vo funkcii `onDraw` a používa niekoľko vykresľovacích priechodov. Detailný postup v jednotlivých krokoch, bude popísaný v nasledujúcich sekciách. Grafický výstup jednotlivý priechodov a ich prepojenie je zobrazené na obrázku nižšie. Postupnosť krokov je nasledujúca:

1. Ak bol detekovaný dotyk, tak sa algoritmom *color picking* určí vybraný objekt v scéne
2. Vykresli sa tieňová mapa do textúry.
3. Vykreslí sa doska a figúrky do textúry v plnom rozlíšení. Súčasne sú vykreslené tiene s využitím tieňovej mapy z predchádzajúceho kroku. Do alfa kanálu je uložená požadovaná úroveň rozostrenia.
4. Textúra z kroku 3 sa zmenší na štvrtinu a vykreslí do textúry

[19] Na platforme Android, pri použití `GLSurfaceView`, dochádza k uvoľneniu OpenGL zdrojov pri strate kontextu (http://developer.android.com/reference/android/opengl/GLSurfaceView.Renderer.html).



Obrázok 21 Priechody pri vykreslení snímku.

5. Textúra z kroku 4 sa rozmaže vertikálne pomocou *Gussovho filtra*.
6. Textúra z kroku 5 sa rozmaže horizontálne.
7. Aplikuje sa *Bloom* a *Depth of Field* efekt pomocou textúr z krokov 3 a 6. Snímok sa vykreslí na obrazovku.
8. V prípade potreby, sa vykresli v rohu obrazovky tieňová mapa a mapa úrovne rozostrenia.

7.4.3 Zobrazenie hracej plochy a figúrok

V scéne sa nachádza jedna kamera charakterizovaná triedou `Camera`. Jej úlohou je poskytnutie pohľadovej a projekčnej matice. Objekt kamery sa vytvára pomocou inštancie triedy `CameraChooser`, ktorá nastaví parametre kamery podľa pomeru strán obrazovky (pri orientácii na šírku je kamera bližšie k hracej ploche, aby sa efektívnejšie zabral priestor scény). Vzniknutá kamera sa dá posúvať a otáčať nastavovaním jej atribútov.

Scéna obsahuje objekt hracej plochy a zoznam políčok. Políčka sa nevykresľujú, ale môžu obsahovať hraciu figúrku. Políčko je definované Triedou `Field`, ktorá obsahuje metódy na manipuláciu s figúrkou, ktorá na ňom stojí. Vykreslenie figúrok potom prebieha tak, že sa prechádzajú všetky políčka, a ak políčko obsahuje figúrku, tak sa vykreslí. Veľkosť vykreslenej figúrky, a teda aj políčka, je závislá od počtu políčok. Čím je viac políčok, tým sú menšie.

V engine je udržiavaná mapa typov figúrok, indexovaná podľa názvu typu. Typy figúrok sú definované v konfiguračnom súbore. Grafický engine obsahuje triedu `FigureType`, ktorá reprezentuje typ figúrky, teda popisuje to, aké má figúrka stavy (figúrka môže mať ľubovoľný počet stavov, ktoré sú popísané v konfiguračnom súbore). Obsahuje dôležitú metódu `createInstance`, ktorá vracia objekt konkrétnej figúrky (trieda `Figure`). Tento objekt, ak je priradený políčku, tak bude vykreslený. Jej vzhľad je odvodený od `FigureType` a je schopná sa vykresliť volaním sekvencie príkazov:

```
1. fig.bindMaterial();// naviaže shader program, podla stavu
2. fig.bindGeometry();// naviaže vertex buffer geometrie
3. fig.draw(); // vykreslí figúrku
```

Okrem toho obsahuje metódu `setState`, pre zmenu stavu figúrky.

7.4.4 Ovládanie

Interakcia užívateľa, v našej aplikácií, prebieha pomocou dotykových gest, práce so vstavaným GUI a podporou pre jednoduchý výber herného políčka.

Dotykové gestá

Podpora pre detekciu dotykových gest je vstavaná do Android frameworku. Detekcia gest je implementovaná vo forme detektorov gest, ktorým sa predáva Android kontext spoločne s poslucháčom, ktorý obsahuje reakciu na dané gesto. V našej aplikácii boli použité detektory `GestureDetector`, `ScaleGestureDetector` a `RotateGestureDetector`[20]. `GestureDetector` zisťuje jednoduché gestá, ako gesto dotyku a gesto posunu – *scroll*. Gesto posunu obsahuje posun v súradniciach x a y voči poslednému stavu. Táto informácia je poslaná

[20] Nie je súčasťou Android API. Implementácia prevzatá od pána Almer Thie zo stránky https://github.com/Almeros/android-gesture-detectors

do grafického enginu a spôsobuje posun hernej dosky v priestore. `ScaleGestureDetector` nám zaisťuje reakciu na gesto rotácie, vďaka čomu zistíme uhol zmeny rotácie oproti predchádzajúcemu stavu. V našej aplikácií spôsobuje rotáciu hernej dosky okolo zvislej osy. Posledným detektorom je `ScaleGestureDetector`, ktorý dovoľuje zistiť faktor zväčšenia a v našej aplikácii plní úlohu pre priblíženia a oddialenia hernej dosky.

GUI

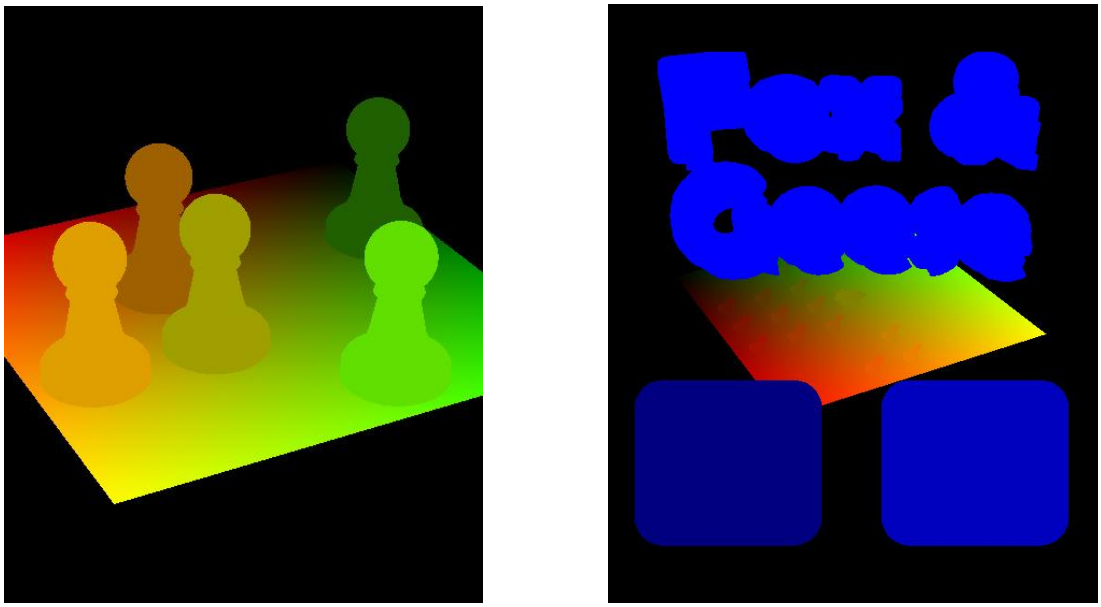
Implementovaný zásuvný modul používa niekoľko spôsobov, akými hráč manipuluje, priamo aj nepriamo, s hernou doskou. Herná logika môže zobraziť obrázky nad hernú plochu, ktoré môžu slúžiť ako tlačidlá. Typy obrázkov sa definujú v konfiguračnom súbore a herná logika potom, podľa potreby, zobrazuje inštancie týchto typov. Vďaka tomu sa dá vytvoriť komplexné užívateľské rozhranie, s rôznymi ponukami. Zobrazený obrázok je reprezentovaný v kóde objektom typu `ImageRencangleInstance`. Taký objekt vzniká volaním metódy `createInstance` nad objektom typu `ImageRectangleType`, ktorý prezentuje typ zobrazovaného obrázka a obsahuje referenciu na textúru a hodnotu `zIndex`, určujúca prioritu prekrývajúcich sa obrázkov.

Výber objektu

Zameranie našej aplikácie má jasné požiadavky na výber objektu v scéne. Je potrebné byť informovaný o dotyku na hracie pole, prípadne na figúrku, ktorá na hracom poli stojí a o dotyku na prvok GUI. Ak nebol dotyk smerovaný ani na figúrku, ani na políčko, ani GUI, tak túto udalosť taktiež potrebujeme detekovať.

Výber objektu pozostáva z niekoľkých fáz a používa *Color picking* algoritmus. V prvok kroku je udalosť dotyku zaznamenaná v Java strane frameworku a preposlaná cez JNI. Sú prijaté súradnice dotyku na obrazovke v pixeloch. V grafickom engine je nastavený príznak o prijatej udalosti a vlákno čaká na výsledok. Udalosť dotyku bola získaná vo vlákne, ktoré patrí užívateľskému rozhraniu, a preto nemôže vykonať výpočet priamo. Výpočet musí prebehnúť vo vlákne, ktoré vlastní OpenGL kontext. Algoritmus je vykonaný pri vykresľovaní nasledujúceho snímku a po získaní výsledku je pomocou synchronizačného primitíva `condition_variable` oboznámené vlákno užívateľského rozhrania, ktoré následne vyvolá jednu z udalostí `onRectangleTouch`, `onFieldTouch` alebo `onEmptyTouch` v hernej logike.

V rámci algoritmu Color picking sme vhodne zakódovali pomocou farieb všetky herné prvky (Obrázok 22). Figúrky a políčka, na ktorých stoja, majú rovnakú farbu. Výsledkom zvoleného prístupu je presný výber objektu v scéne a jednoduché pridanie podpory pre detekciu dotykov na GUI. Zvolený prístup je, z výkonnostného hľadiska, dostačujúci pre tento typ hier.



Obrázok 22 Farebná mapa algoritmu Color picking. Vľavo farebne odlíšené figúrky podľa políčka na ktorom stoja. Vpravo, v modrých odtieňoch, sú vyznačené prvky GUI.

7.4.5 Gama korekcia

Gama korekcia je jedným z najjednoduchších a najlacnejších úprav, ktoré sa dajú použiť na zvýšenie kvality aplikácie. Zachytávanie (fotografovanie, skenovanie) a zobrazovanie obrazu je nelineárny proces [18]. Nelinearita ovplyvňuje výsledok vykresľovania, no stačia jednoduché kroky na nápravu. Pri výpočtoch v shaderoch sa predpokladá, že vstupné dáta sú lineárne, pretože výpočty vykonané v nich sú tiež lineárne. Je však pravdepodobné, že dodané textúry majú aplikovanú gama korekciu. Pri lineárnych výpočtoch nad nelineárnymi dátami môže dochádzať k artefaktom. Správnym postupom je vykonať korekciu do lineárneho priestoru napríklad umocnením hodnoty[21]. Druhým krokom je aplikovať gama korekciu tesne pred zobrazením výstupu na obrazovku. Prevod do gama priestoru sa dá vykonať odmocnením hodnoty.

V našom vykresľovaní dochádza k linearizácii pri použití textúr v materiáloch, pretože nad nimi prebiehajú výpočty osvetlenia, a ku gama korekcii pri zobrazení scény na obrazovku v poslednom priechode. Pri zobrazovaní GUI sa s gama korekcia nepoužíva, pretože nedochádza k výpočtom pri ich zobrazovaní.

```
...
// linearizacia (predpokladane gammu 2.0 namiesto 2.2)
lowp vec3 color = texture2D(colorTexture, f_uv).rgb;
color = pow(color, vec3(2.0));
...
// gamma korekcia (predpokladame gammu 2.0 namiesto 2.2)
color = sqrt(color);
gl_FragData[0] = vec4(color);
...
```

Obrázok 23 Ukážka lacnej linearizácie a gama korekcie v GLSL shadery.

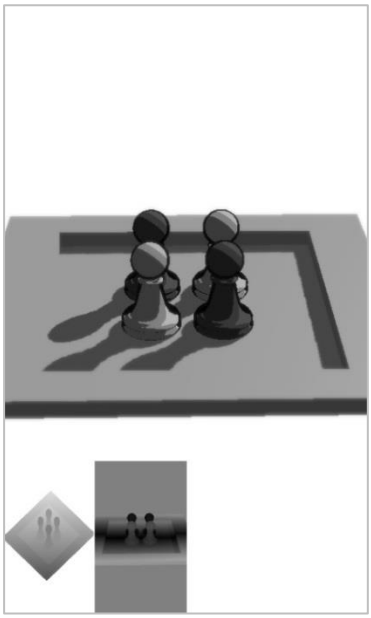
[21] Odporúčaným postupom je použiť sRGB formát textúr, ale na OpenGL ES 2.0 nie je dostupný (16).

7.4.6 Grafické efekty

Na implementáciu grafických efektov sa osvedčila trieda `FrameBuffer`. Vďaka nej bola tvorba efektov jednoduchšia a menej náchylná na chyby. Grafické efekty vo svojich priechodoch potrebovali framebuffre s pripojeným *color bufferom*. Niektoré využili pripojenie *depth buffera*.

Ladenie grafických efektov

Grafické efekty sú väčšinou implementované viacerými priechodmi. Výstup je jedného priechodu prichádza na vstup iného. Problém s vytváraním grafických efektov je, že stav grafického reťazca je ťažko pozorovateľný, neexistujú ladiace výpisy alebo krokovanie v shader programoch. Preto na zobrazenie ladiacich informácii bol zvolený grafický ladiaci výpis. To znamená že na obrazovku je okrem samotnej scény vykreslený aj obsah výstupov jednotlivých priechodov (Obrázok 24).



Obrázok 24 Znázornenie tieňovej mapy a mapy intenzity rozostrenia v ľavom dolnom rohu obrázovky.

Efekt hĺbky ostrosti a Bloom efekt

Efekt hĺbky ostrosti aj Bloom efekt sú techniky, ktoré vyžadujú rozostrenie obrazu. V našom prístupe sme sa rozhodli rozostrenie vykonať iba raz, pre oba grafické efekty. Tieto efekty sa potom líšia iba v spôsobe výpočtu úrovne rozostrenia.

Úroveň rozostrenia je počítaná v prvom priechode a ukladá sa do alfa kanálu[22]. Samozrejme je nevýhodou, že nie sme schopný vykresliť priehľadné objekty, no vďaka tomuto spôsobu riešenia sme ušetrili jeden grafický priechod. V druhom priechode, sa výstup z prvého priechodu vykreslí do textúry, s polovičnými rozmermi strán. Je to prvý krok rozostrenia, pretože na konci pri zlučovaní s textúrou v plnom rozlíšení, bude musieť byť táto textúra zväčšená, a tým sa ešte rozmaže. Hlavným

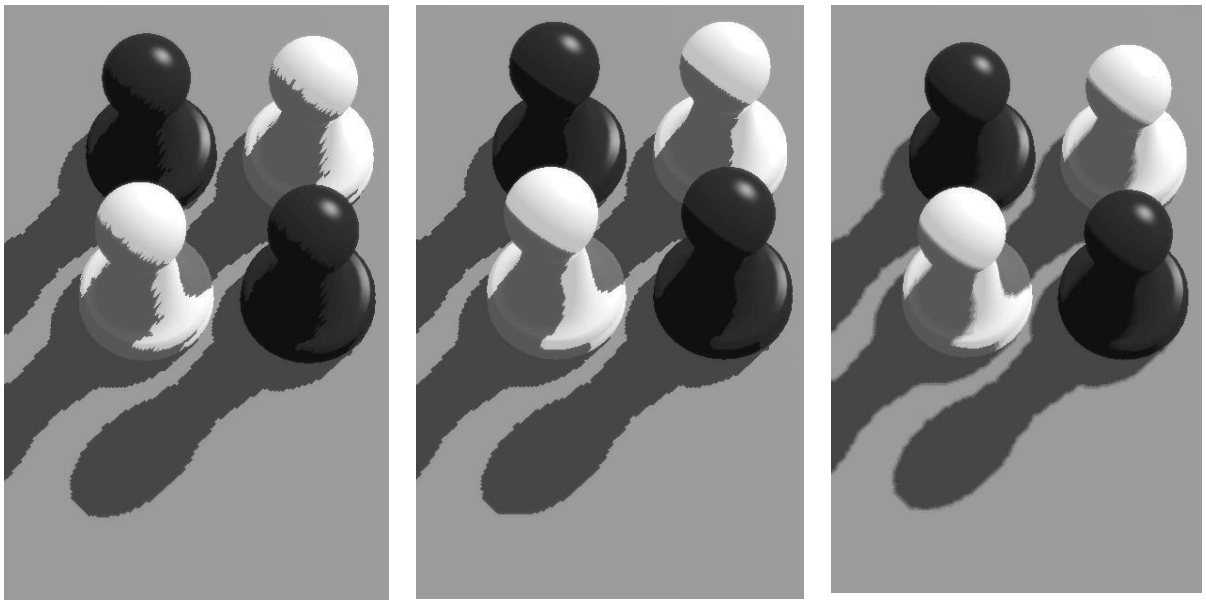
[22] V OpenGL ES 2.0 nie je možné vykresľovať do viacerých cieľom v jednom priechode [20].

dôvodom ale je zmenšenie plochy, ktorú musí rozmazať Gaussov filter. Gaussov filter je aplikovaný v dvoch priechodoch, vertikálne a horizontálne. Je použité jadro o veľkosti 5x5 a aby sme predišli počítaniu textúrovacích súradníc vo fragment shadery, boli tieto súradnice predpočítané vo vertex shadery a následne interpolované pred použitím vo fragment shadery. V úplne poslednom priechode je spojený výstup z prvého a predposledného priechodu (rozmazaná textúra) podľa hodnoty alfa kanálu z predposledného priechodu.

Úroveň Bloom efektu je odvodená od intenzity spekulárnych odleskov na povrchu objektu a úroveň rozostrenia pri efekte hĺbky ostrosti je závislá od vzdialenosti od bodu zaostrenia, ktorý je umiestnený v strede hracej dosky.

Tieňové mapy

Súčasťou nášho vykresľovacieho enginu je algoritmus tieňových máp. V scéne vyobrazuje tieňe vrhané figúrkami alebo hernou doskou. Rozlíšenie tieňovej mapy je štvrtinové oproti rozlíšeniu obrazovky, čo je kompenzované vyhladzovaním hrán algoritmom PCF. Pri vyhladzovaní hrán sa čítajú štyri hodnoty z tieňovej mapy namiesto jednej. Pozície UV koordinát sú predpočítanie



Obrázok 25 Tieňová mapa. Vľavo sú artefakty na oblých povrchoch, v strede je možno vidieť ich odstránenie pomocou variabilnej hodnoty *bias* a v pravo sú tiene vyhladené technikou PCF.

vo vertex shadery a ich pozície vychádzajú z Poisonovho disku. Každý koordinát má pri výpočte tieňa rovnakú váhu. Na odstránenie efektu akné bola použitá hodnota *bias* (Obrázok 25). Bias hodnota posúva z súradnicu fragmentu v súradniciach svetla, pre ktorý počítame hodnotu tieňa, ďalej do priestoru, aby nevznikal *z-fight* medzi ním a tieňovou mapou. Problem nastáva pri pevne nastavenej hodnote bias na zahnutých povrchoch. Vzniknuté artefakty sme potlačili výpočtom hodnoty bias na základe sklonu povrchu.

Algoritmus vo svojom prvom priechode vykresľuje tieňovú mapu do textúry typu float, no knižnica OpenGL ES 2.0 taký typ textúr neposkytuje. Preto bolo nutné zakódovať float hodnoty

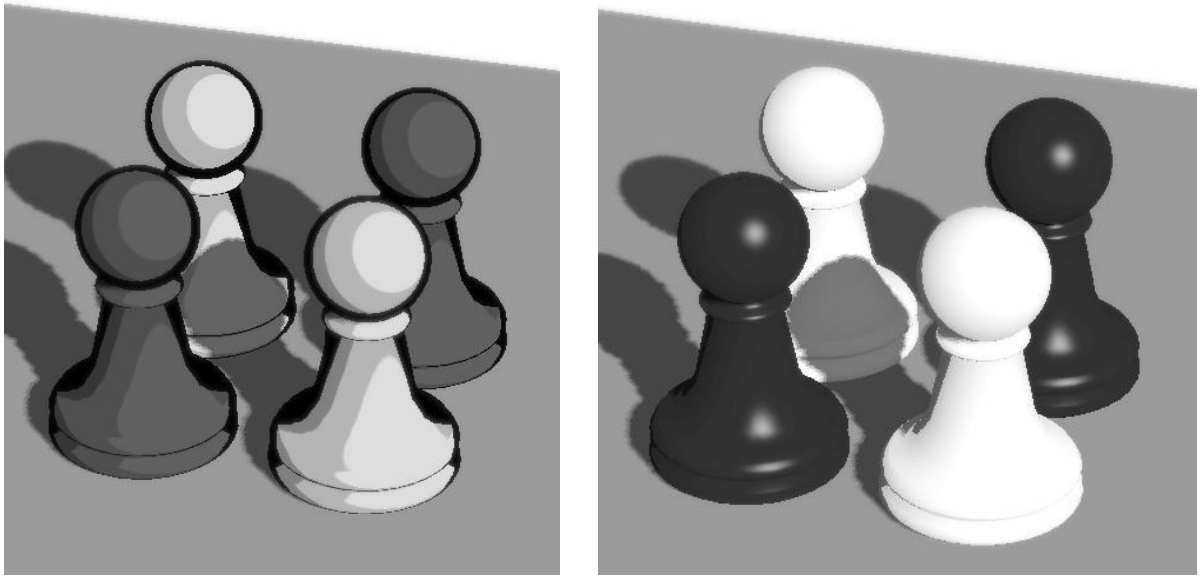
do formátu RGBA a potom pri použití znova rozkódovať. Pre tento účel vznikli GLSL funkcie `pack_float` a `unpack_float`. Iným prístupom je použitie OpenGL rozšírenia `EXT_color_buffer_half_float`.

Práca s tieňovými mapami musí byť integrovaná do každého materiálu, aby bolo možné zobraziť tiene vrhané na objekt s daným materiálom. Preto vznikli GLSL pomocné funkcie, ktoré dovoľujú jednoduché zapojenie podpory pre tiene do každého nového materiálu (viac v 7.4.7).

7.4.7 Manipulácia s materiálmi

Grafický engine obsahuje niekoľko preddefinovaných typov materiálov. V konfigurácii zásuvného modulu sa potom vytvárajú konkrétne výskyty daného typu materiálu. Konkrétny výskyt je potom použitý, aby určil vzhľad prvkov v scéne a je definovaný triedou `Material`. Na konkrétny výskyt sa potom odkazuje cez jeho meno. Priebeh tvorby konkrétneho výskytu, z typu materiálu je nasledujúci: Vytvorí sa objekt typu Material, pričom ako parameter konštruktoru sa mu zadá lambda výraz. Predaný lambda výraz obsahuje chovanie materiálu pri jeho použití. Lambda výraz dostane objekt triedy `ShaderProgram` a v capture zozname[23] parametre špecifické pre daný typ materiálu (napríklad pre *SolidMaterial* sú to intenzita odleskov a textúra povrchu). Okrem toho má prístup ku objektu typu `MaterialContext`, ktorý je globálnou premennou modulu materiálov. `MaterialContext` dodáva do materiálu matice, informácie o svetlách a informácie o efektoch.

Každý typ materiálu je okrem spomínanej lambdy určený aj svojím shader programom. Shader programy obsahujú kód shaderov, ktorý je vlastný pre daný typ materiálu, ale aj kód, ktorý sa opakuje vo všetkých materiáloch. Takým zdieľaným kódom, je napríklad kód grafických efektov. Znovu



Obrázok 26 ToonMaterial vľavo a SolidMateial vpravo.

[23] Spôsob, akým sa lambde sprístupňujú vonkajšie entity [19].

použiteľnosť kódu je implementovaná jednoducho, a to spájaním reťazcov, a potom volaním funkcií, ktoré boli definované v zdieľanom fragmente kódu. V súbore `shaderSourceFragment.h` sú práve takéto zdieľané fragmenty kódu.

SolidMaterial

Typ materialu nazvaný *SolidMateriál*, je základným typom materiálu. Objekty vykresľuje pomocou Phongovho osvetľovacieho modelu a Phongovho tieňovania. Farba povrchu je závislá od dodanej textúry a od UV koordinát objektu. Intenzita lesku objektu je ovplyvniteľná parametrom *glossines.*

ToonMaterial

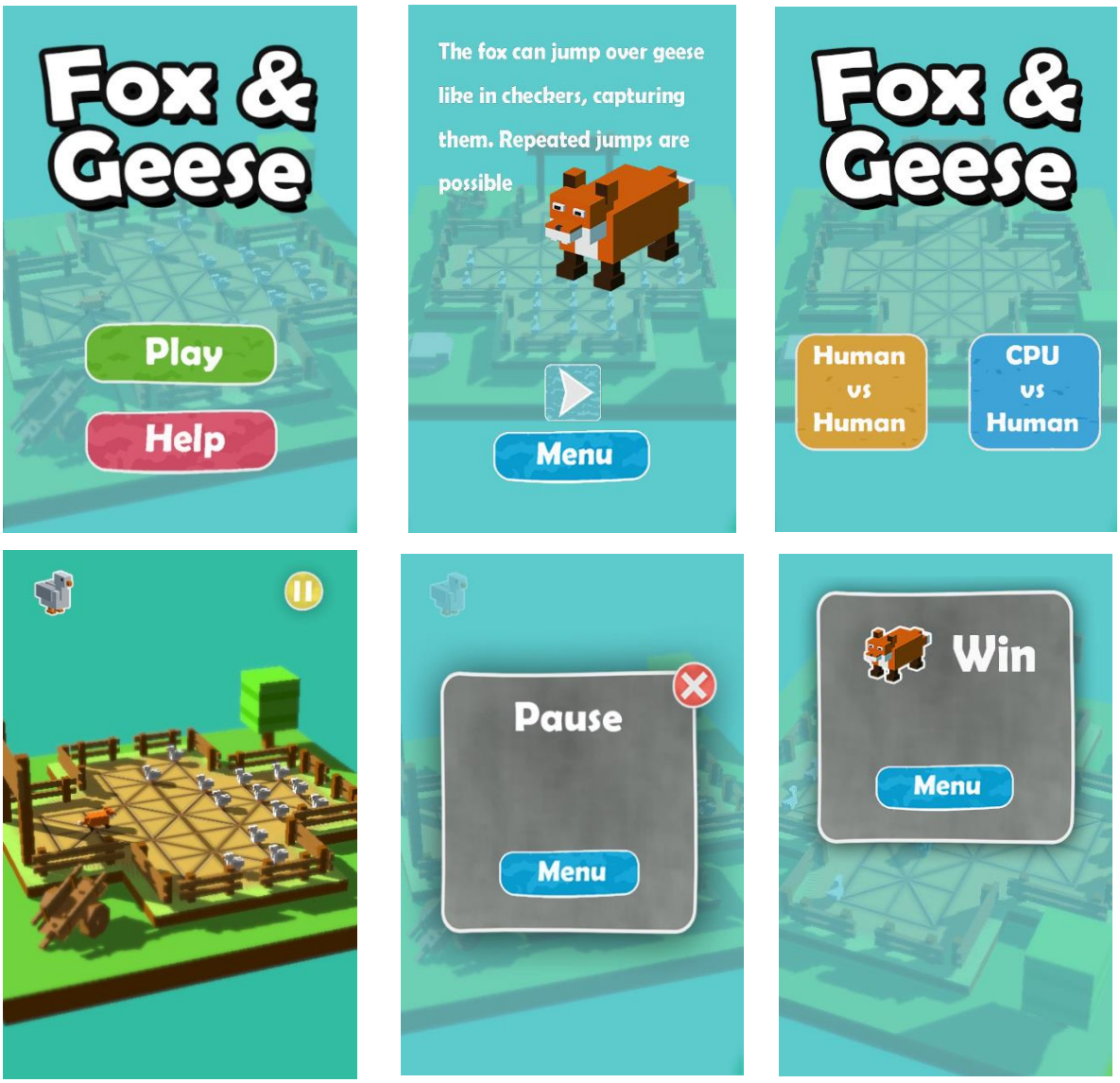
Ďalším typom je *ToonMaterial*, ktorý vykresľuje objekty ako kreslené. Je parametrizovateľný základnou farbou, podobne ako u SolidMaterialu, a 1D *ramp* textúrou. Práve ramp textúra ovplyvňuje, aké a koľko bude mať kreslený material farebných úrovní. Jednotlivé pixli obsahujú farbu v stupňoch šedej, a tým určuje prah pri výpočte farebných odtieňov. Počet farebných odtieňov je zadaný ako parameter v konfigurácií. Okrem zníženia počtu odtieňov sa objektu pridáva silueta. ToonMaterial je vhodný hlavne pre detailné hladké povrchy, v opačnom prípade môžu vznikať artefakty kvôli efektu siluety. Ukážku použitia je možné vidieť na obrázku vyššie.

7.5 Implementovaná dosková hra

Na otestovanie nášho enginu sme si zvoli doskovú hru Líška a husi – *Fox & Geese.* Jedná sa o anglickú variáciu na hru Vlk a ovce (2.1.2). V nasledujúcich sekciách si popíšeme vytvorenie grafických prvkov, hernej logiky, umelej inteligencie a užívateľského rozhrania.

7.5.1 Grafické prvky

Na vytvorenie geometrie bol použitý program Blender. Geometria objektov vychádza z kocky, na ktorú boli hlavne použité operácia *extrude* a *modifikátor boolean*, vďaka čomu vznikol ich špecifický vzhľad. Vrcholom modelovaných objektov bola nastavená farba, ktorá bola exportovaná v podobe farebnej mapy. Povrchu modelov boli nastavené UV súradnice. Takto bol vytvorený model pre hernú dosku, líšku a pre hus. Obrázkové prvky, napríklad tlačidlá, pomoc, dialógové boxy, logo a stavové obrázky, vznikli pomocou grafického editora. Zložitejšie prvky užívateľského rozhrania, ako dialógové okná, vznikli vrstvením obrázkov so zadanou hodnotou *zIndex*.



Obrázok 27 Vzhľad hry Fox & Geese. Postupne zľava doprava, zhora dole: Úvodná obrazovka, obrazovka pomocníka, voľba protihráča, rozohraná partia, dialógové okno pozastavenia hry, koniec hry – obrazovka víťaza.

7.5.2 Umelá inteligencia

Pri spustení, si hráč môže zvoliť za svojho protihráča umelú inteligenciu. Umelá inteligencia môže ovládať líšku alebo husi. Ako základ bol zvolený algoritmus **Minimax**, ktorý používa prehľadávanie stavového priestoru do hĺbky, pričom stavy ohodnocuje pomocou hodnotiacej funkcie. Hodnotiaca funkcia berie do úvahy počet husí, postavenie figúrok (pozícia na diagonále) a obkľúčenie líšky. Husi sa snažia, o čo najvyššie ohodnotenie a líška o čo najnižšie. Stav je charakterizovaný zoznamom políčok (informácia o obsadenosti), zoznamom pozícii husí a pozíciou líšky. Aby sme zaručili ukončenie algoritmu v rozumnom čase, je zanorenie algoritmu je obmedzené hĺbkou 3.

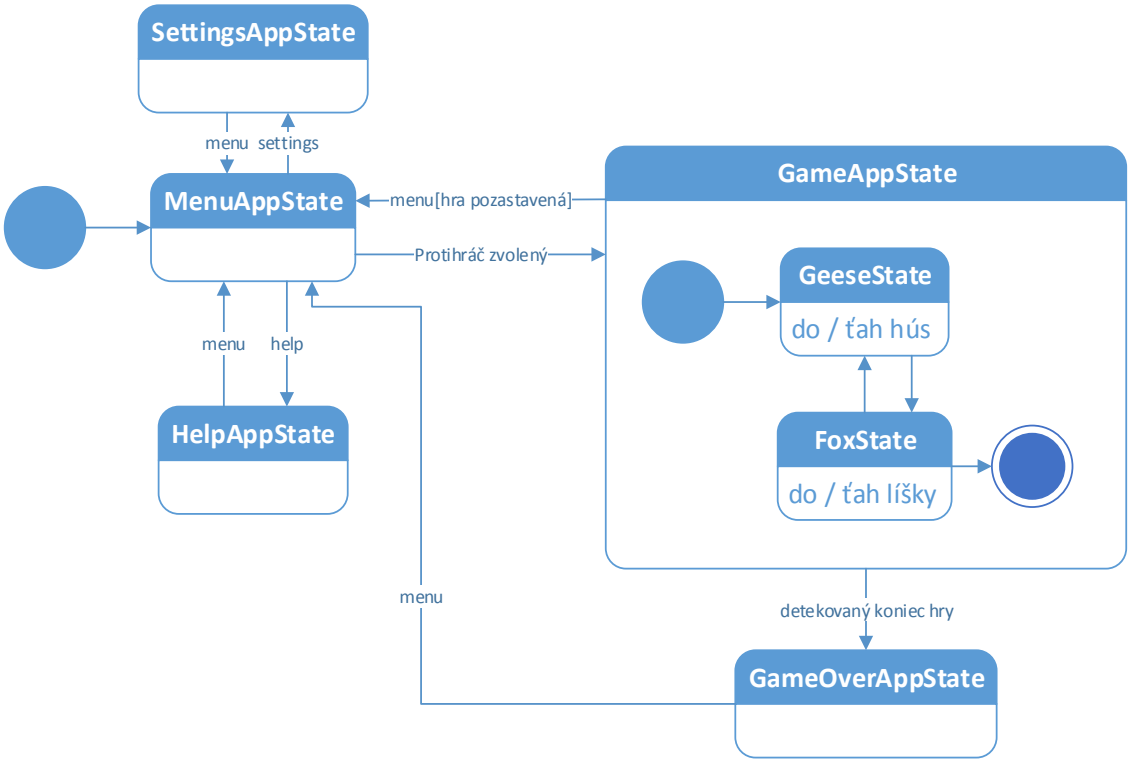
7.5.3 Herná logika

Herná logika je na najvyššej úrovni reprezentovaná návrhovým vzorom Stav – *state*. Používa dva zanorené stavové automaty (Obrázok 28). Jedným je automat stavu aplikácie, ktorý prezentuje jednotlivé obrazovky hry. Druhým je automat samotnej hernej partie, ktorý je vnorený do stavu `GameAppState`. `GameAppState` pozostáva zo stavov `GeeseState` a `FoxState`, teda ťahy oboch strán, ktoré vykonávajú ťah a kontrolujú koniec hry. Všetky stavy sú potomkami typu `AppState`, a ich úlohou je reakcia na udalosti, zobrazenie GUI, uloženie a obnovenie svojho stavu. Každý stav je jednoznačne identifikovateľný pomocou svojho ID. Pri ukončený aplikácie sa ukladá aj informácia o aktuálnom stave, aby pri obnovení aplikácie sa mohlo pokračovať práve z toho stavu. Návrh aplikácie pomocou stavov je výhodný z pohľadu jednoduchej rozšíriteľnosti aplikácie.

Figúrky sa môžu nachádzať v dvoch stavoch – *normal* a *selected*, medzi ktorými sa prechádza pri ťahu. Pred vykonaním ťahu je nutné figúrku označiť a potom zvoliť cieľovú pozíciu. Výber políčka je znázornení jeho podfarbením na krátku dobu.

7.5.4 Užívateľské rozhranie

Užívateľské rozhranie pozostáva z niekoľkých obrazoviek, medzi ktorými sa prechádza (Obrázok 27). Na niektorých prvkoch GUI boli použité animácie, pomocou časovača. Jednoduchá podpora pre animácie bola vložená do modulu `animation`.



Obrázok 28 Stavový diagram hry Fox & Geese.

8 Zhodnotenie dosiahnutých výsledkov

Výsledkom práce je knižnica, ktorá po pripojení do projektu, umožňuje vytvorenie doskovej hry. Takto vytvorená dosková hra môže strategická, závodná alebo aj pozičná. Dosková hra pozostáva z Android Fragmentu, XML konfiguračného súboru, hernej logiky v jazyku ECMAScript a podporných súborov (obrázky, geometria). Pri návrhu bol zvolený prístup, ktorý umožnil, aby programátor bol odstienený od nutnosti práce s Android API. Vďaka skúsenostiam, ktoré sa nadobudli pri tvorbe doskovej hry Fox & Geese, s využitím našej knižnice, môžeme zhodnotiť zvolený prístup. Najväčší čas sme strávili pri budovaní hernej logiky, konkrétne pri implementácii pravidiel a umelej inteligencie. Tým, že herná logika bola implementovaná vo vysokoúrovňovom programovacom jazyku, bolo umožnené, aby vývoj postupoval relatívne rýchlo. Poskytnuté doménové API dovolilo ľahkú manipuláciu s objektmi v scéne. Jazyk ECMAScript, v kombinácii s interaktívnym príkazovým riadkom, umožnil prototypovanie nápadov. Budovanie grafického užívateľského rozhrania, taktiež bolo rýchle a nevyžadovalo veľký počet riadkov zdrojového kódu.

Deklaratívna definícia zásuvného modulu pomocou XML formátu, sa ukázala ako správna, z niekoľkých dôvodov. Poskytnuté API v hernej logike nemusí obsahovať funkcie, ktoré nastavujú nemenné parametre. Cesty k pomocným súborom nemusia byť napevno zapísané v hernej logike. Ďalšou výhodou je jednoduchá rozšíriteľnosť, úpravou XML schémy.

Vstavané grafické efekty zvýšili príťažlivosť scény. Ukázalo sa, že výkonnosť aktuálnych mobilných zariadený, je dostatočná na aplikovanie grafických efektov, akými sú tieňové mapy s vyhladzovaním okrajov, bloom efekt a efekt hĺbky ostrosti v reálnom čase. I keď, sú dostupné vyššie verzie grafickej knižnice OpenGL, my sme sa rozhodli pre verziu 2.0, vďaka čomu, náš framework možno spustiť na väčšine Android zariadení.

V nasledujúcich sekciách si ukážeme, ako obstojí náš framework, v podobe hry Fox & Geese, na rôznych zariadeniach. Potom ho porovnáme s inými možnými prístupmi a nakoniec budeme diskutovať možnosti ďalšieho vývoja.

8.1 Porovnanie s inými prístupmi

Porovnateľnú funkcionalitu, ktorú ponúka náš vykresľovací engine, je možné dosiahnuť rôznymi spôsobmi. My teraz porovnáme náročnosť na implementáciu doskovej hry, inými prístupmi, ako s naším frameworkom.

Na zelenej lúke

V tomto prípade, by musela byť napísaná celá aplikácia, iba s využitím komponent, ktoré ponúka platforma Android. Celá implementácia, by mohla byť Jave, a teda by sa nemuselo

komunikovať cez JNI. Na grafické užívateľské rozhranie, by sa mohli použiť vstavané Android prvky. Jednotlivé obrazovky hry, by boli samostatné aktivity, medzi ktorými by sa prechádzalo. Odhadovaný čas implementácie by bol podstatne vyšší, ako s naším frameworkom, pretože by muselo byť samostatne vytvorené vykresľovcie jadro, spolu s efektmi a herná logika by bola implementovaná tiež Jave, čo nie je tak efektívne, ako použitie skriptovacieho jazyka.

Použitie herného enginu

V súčasnej dobe, existuje niekoľko kvalitných herných enginov, ktoré dovoľujú vystavanie rôznych typov hier. V krátkosti spomeňme:

Unity[24]

Patrí medzi najpopulárnejšie enginy na mobilných platformách. Ponúka zaujímavé licenčné podmienky pre začínajúcich vývojárov. Obsahuje mnoho nástrojov použiteľných pri tvorbe hier. Samozrejmosťou je možnosť skriptovania.

Unreal Engine 4[25]

Herný engine použitý mnohými modernými, kvalitnými hrami. Množstvo používateľov tvorí rozsiahlu komunitu.

Výhodou herného enginu je, že obsahuje väčšinu prostriedkov, ktoré sú nutné na vytvorenie hry. V základe sú stavané tak, aby sa dali použiť na rôznych platformách. Z nevýhod spomeňme, že krivka učenia herného enginu nemusí byť strmá. V porovnaní s naším frameworkom, je vytvorenie doskovej hry, časovo náročnejšie. Dôvodom je hlavne dlhšia doba učenia a nutnosť implementovať komponenty súvisiace s doskovými hrami (manipulácia s figúrkami, zobrazenie scény).

8.2 Test výkonu

Náš framework na tvorbu doskových hier bol otestovaný na 6 zariadeniach. Test prebiehal na implementovanej hre. Skúmala sa celková funkčnosť a počítal sa počet vykreslených snímkou za sekundu. Testované zariadenia sa líšili v hardwarovej výbave, veľkosti obrazovky a verzií systému Android. Vykresľovacia frekvencia bola počítaná pri rôznych grafických nastaveniach. Vykresľovaná scéna pozostávala z hracej dosky a šestnástich figúrok, čo prestavuje približne 4456 polygónov. Počet snímkou za sekundu bol počítaný v trojsekundových intervaloch. V tabuľke nižšie, sú FPS hodnoty pre rôzne efekty.

[24] https://unity3d.com/unity
[25] https://www.unrealengine.com/what-is-unreal-engine-4

Zariadenie	Rok uvedenia	Bez efektov	Tieňové mapy	Tieňové mapy PCF	Efekty rozmazania	Všetky efekty
Sony Xperia L	2013	60.00 fps	60.00 fps	60.00 fps	60.00 fps	60.00 fps
Nexus 7 (2012)	2012	30.00 fps	24.66 fps	24.66 fps	25.33 fps	21.66 fps
Samsung Galaxy Nexus	2011	15.00 fps	10.00 fps	9.33 fps	11.66 fps	8.00 fps
Nexus 10	2012	28.33 fps	19.33 fps	16.66 fps	22.33 fps	14.33 fps
Acer Iconia Tab 8A1-840FHD	2014	53.66 fps	45.66 fps	43.33 fps	41.66 fps	40.33 fps
Nvidia Shield Tablet	2014	60.00 fps	60.00 fps	60.00 fps	60.00 fps	60.00 fps

Tabuľka 3 FPS hodnoty pre rôzne spustené efekty. Efekt rozmazanie predstavuje Depth of Field efekt a Bloom efekt. Tieto dva efekty sa spúšťajú a vypínajú súčasne. Hodnoty sú v jednotkách *snímky za sekundu*. Hodnota 60 fps je maximálna hodnota vykresľovacej frekvencie na platforme Android.

Ako možno vidieť, najnižšie hodnoty boli zaznamenané na zariadení Samsung Galaxy Nexus, no je nutné poznamenať, že rok predstavenia zariadenia je 2011. Na aktuálnych a budúcich zariadeniach by nemal byť problém, spustiť aplikáciu so všetkými efektmi pri slušnom FPS.

8.3 Budúci vývoj

Náš systém na tvorbu doskových hier by sa mohol vylepšiť v mnohých oblastiach. Množina ovládacích prvkov je obmedzená. Bolo by vhodné pridať aspoň vykresľovanie textu s voliteľnými typmi písem, vstupné textové polia alebo iné spôsoby rozloženia prvkov. Ďalej by bolo vhodné, poskytnúť podporu pre rôzne rozlíšenia obrázkov, ktoré by sa vyberali podľa hustoty pixelov na zariadení. Keďže systém na tvorbu užívateľského rozhrania môže byť značne komplexný, tak vhodnou voľbou sa javí integrácia už existujúcej knižnice.

Momentálne, vo frameworku neexistuje možnosť, ako umiestniť figúrky mimo hracie políčka. Ak by existovala, a figúrkam by dala nastavovať pozícia, rotácia a zväčšenie, bolo by možné s využitím časovača, implementovať animácie figúrok. V spolupráci s ECMAScriptom, by sa otvorili široké možnosti použitia. Do skriptovacie jazyka by bolo vhodné pripojiť aj ďalšie API na ovládanie grafického enginu. Mohlo by pribudnúť podrobnejšie nastavenie grafických efektov (rozlíšenia framebufferov), či pohyb kamery a scény.

Podstatná časť systému je vytvorená v jazyku C++ a je teda platformovo nezávislá. Ďalšou výzvou by bolo, preniesť framework aj na iné platformy. Kandidátmi sú hlavne iOS a Windows[26]. Niektoré časti systému by sa museli presunúť do natívnej časti kódu (načítanie XML, spracovanie dotykov) a iné by museli byť platformovo závislé (ukladanie dát, práca so súbormi).

[26] Je myslený systém Windows 10, ktorého aplikácie by mali fungovať od desktopov, po mobilné zariadenia. Bolo by však nutné, podporovať vykresľovanie aj cez rozhranie DirectX.

9 Záver

V našej práci sme preskúmali rôzne doskové hry, z čoho sme mohli spresniť požiadavky na výsledný systém. Vybrali sme si dve doskové hry, ktorým sme navrhli prechody medzi obrazovkami, vzhľad 3D grafického rozhrania a spôsob ovládania. Potom sme preskúmali možnosti implementácie v natívnom kóde na platforme Android, čo nám umožní efektívnejšiu prácu s grafikou a prenositeľný kód. V práci sme zaoberali aj grafickými efektmi. Zamerali sme sa na princípy a na spôsob ich implementácie v OpenGL. Bližšie sme sa pozreli na filtrovacie algoritmy, ktoré sú súčasťou mnohých efektov.

Hlavným prínosom práce je vytvorenie systému na efektívne vytváranie doskových hier. Klient používajúci náš systém ho môže jednoduchým spôsobom integrovať a vytvoriť doskovú hru pomocou konfigurácie napísanej v jazyku XML a hernej logiky napísanej v skriptovacom jazyku ECMAScript. Pri tvorbe zásuvného modulu klient nemusí pracovať s Android API. Na použitie sú pripravené grafické efekty ako tieňové mapy, hĺbka ostrosti alebo bloom. Náš systém bol vyskúšaný pri implementácií doskovej hry *Fox & Geese*. Vytvorená hra bola otestovaná na niekoľkých mobilných zariadeniach a bol zmeraný grafický výkon. Na záver sme diskutovali ďalší vývoj, pričom bol spomenutý prenos na iné platformy, či doplnenie možností 2D užívateľského rozhrania

Literatúra

1. **Zapletal, Miloš.** *Velká kniha deskových her.* Praha : Mladá Fronta, 1991. ISBN 80-204-0188-4.

2. **Pijanowski, Lech a Pijanowski, Wojciech.** *Encyklopedie světových her.* s.l. : Universum, 2008. ISBN 978-80-242-2225-7.

3. **Filip.** An investigation of fast real-time GPU-based image blur algorithms. *Intel Developer Zone.* [Online] 14. 7 2014. [Dátum: 7. 1 2015.] https://software.intel.com/en-us/blogs/2014/07/15/an-investigation-of-fast-real-time-gpu-based-image-blur-algorithms.

4. *Variable-sized, Circular Bokeh Depth of Field Effects.* **Moersch, Johannes and Hamilton, Howard J.** Toronto : Canadian Information Processing Society, 2014, Zv. GI '14, s. 103-107. 978-1-4822-6003-8.

5. *Casting Curved Shadows on Curved Surfaces.* **Williams, Lance.** 3, New York : ACM, august 1978, Vol. 13, pp. 270-274. 0097-8930.

6. *Rendering Antialiased Shadows with Depth Maps.* **Reeves, William T. and Salesin, David H. and Cook, Robert L.** 4, New York : ACM, august 1987, Zv. 21, s. 283-291. 0097-8930.

7. **Mitchell, Jason L.** Poisson Shadow Blur. [aut. knihy] Wolfgang Engel. *ShaderX3 Advanced Rendering with DirectX and OpenGL.* Rockland : Charles River Media, Inc., 2004, 5, s. 403-409.

8. **Fernando, Radima a Kilgard, J. Mark.** *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics.* Boston : Addison-Wesley Longman Publishing Co., 2003. ISBN 9780321194961.

9. **Lipchak, Benjamin, Haemel, Nicolas and Wright, Richard S.** *OpenGL SuperBible Fourth Edition.* Michigan : Addison-Wesley, 2007. ISBN 9780321498823.

10. **Adams, Ansel.** *The Camera.* The Ansel Adams Photography Series. New York : Little, Brown, 1995. s. 48-54. 978-0821221846.

11. **Engel, Wolfgang.** *ShaderX7: Advanced Rendering Techniques.* Boston : Charles River Media, 2009. 978-1584505983.

12. **Henriksson, Ola.** A Depth of Field Algorithm for Realtime 3D Graphics in OpenGL. *DiVA.* [Online] 1 31, 2002. [Cited: 1 7, 2015.] http://www.diva-portal.org/smash/get/diva2:18115/FULLTEXT01.pdf.

13. *X-toon: an extended toon shader.* **Barla, Pascal and Thollot, Joelle and Markosian, Lee.** Annecy : ACM, 2006. Proceedings of the 4th International Symposium on Non-photorealistic Animation and Rendering. s. 127-132. 1-59593-357-3.

14. **Eberly, David H.** *3D Game Engine Architecture: Engineering Real-Time Applications with Wild Magic.* San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2004. 012229064X.

15. **Allen, Grant.** *Android 4: průvodce programováním mobilních aplikací.* Brno : Computer Press, 2013. 978-80-241-3782-6.

16. **Kosek, Jiří.** *PHP a XML.* Praha : Grada Publishing a.s., 2009. 978-80-247-1116-4.

17. ANGLE: Almost Native Graphics Layer Engine . *Project Home.* [Online] 5. január 2015. [Dátum: 4. máj 2015.] https://code.google.com/p/angleproject/.

18. **Google.** Activity. *Android Developers.* [Online] Google, 30. Apríl 2015. [Dátum: 1. Máj 2015.] http://developer.android.com/reference/android/app/Activity.html.

19. —. Fragments. *Android Developers.* [Online] Google. [Dátum: 1. Máj 2015.] http://developer.android.com/guide/components/fragments.html.

20. Simple 2.7.1. *Simple.* [Online] [Dátum: 3. máj 2015.] http://simple.sourceforge.net/.

21. **Nguyen, Hubert.** *Gpu gems 3.* s.l. : Addison-Wesley Professional, 2007. 9780321545428.

22. **Bjarne, Stroustrup.** *A Tour of C++.* Michigan : Addison-Wesley Professional, 2013. 978-0-321-958310.

23. **The Khronos Group Inc.** OpenGL Common Profile Specification Version 2.0.25 (Full Specification). *khronos.org.* [Online] 2. november 2010. [Dátum: 7. máj 2015.] https://www.khronos.org/registry/gles/specs/2.0/es_full_spec_2.0.25.pdf.

Popis skratiek

Skratka	Prepis	Popis
NDK	*Native development kit*	Množina nástrojov na platforme Android dovoľujúca implementáciu niektorých častí programu v natívnom kóde.
SDK	*Software development kit*	Súbor nástrojov umožňujúci tvorbu aplikácii pre danú platformu.
RTTI	*Run-Time Type Information*	Mechanizmus, ktorý poskytuje informácie o typu objektu za behu programu.
STL	*Standard Template Library*	Knižnica pre jazyk C++. Poskytuje bežné triedy ako napr. rôzne typy kontajnerov.
JNI	*Java Native Interface*	Programovací framework jazyka Java, dovoľujúci volať a byť volaný z natívnych aplikácii.
PCF	*Percentage-closer filtering*	Výpočet hodnoty z hodnôt získaných z okolitých bodov pixelu.
FPS	*frames per second*	Jednotka udávajúca počet snímkou za sekundu.

Zoznam príloh

Príloha A Diagram tried

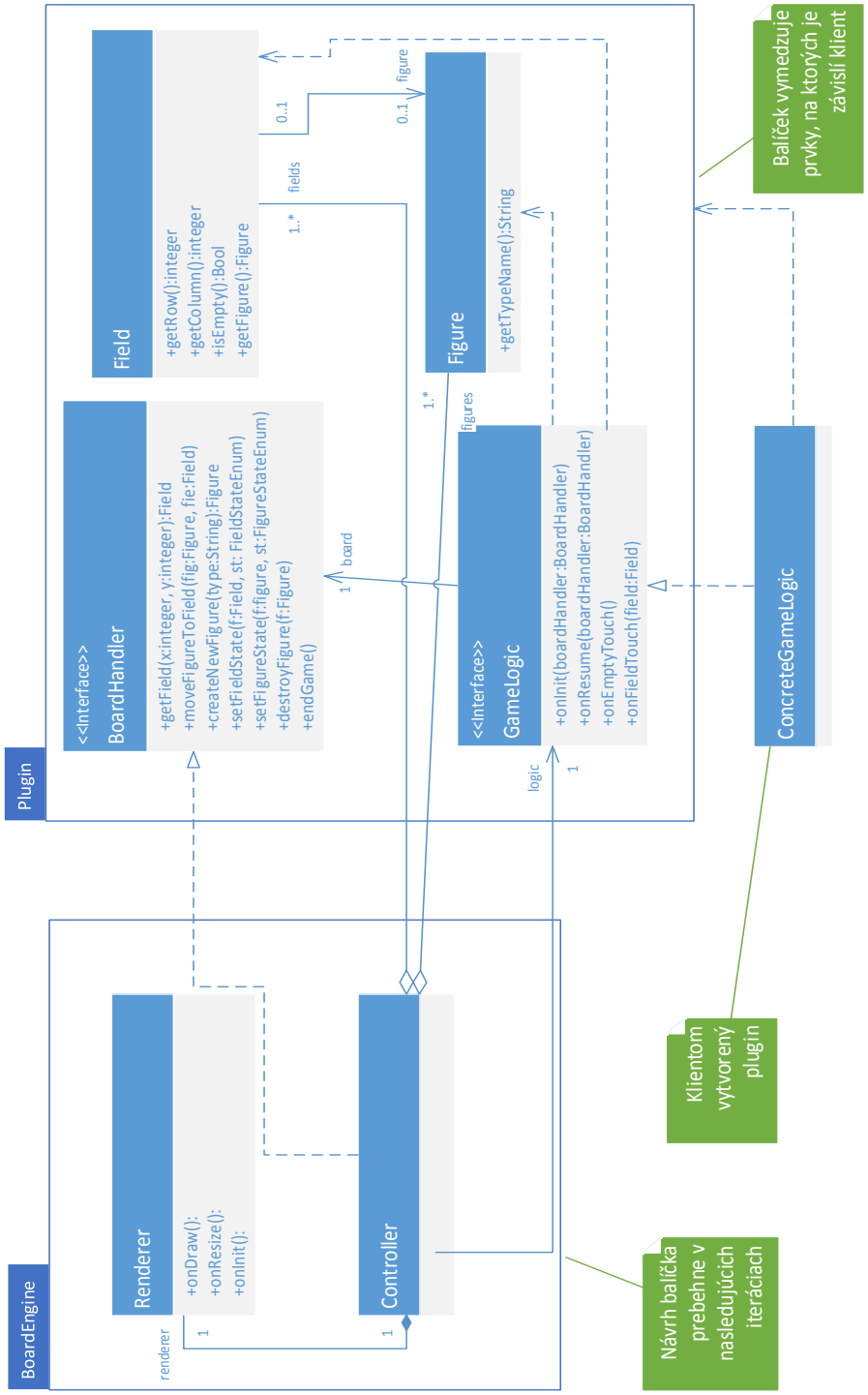
Príloha B Diagram sekvencie

Príloha C Inštalácia

Príloha D Manuál

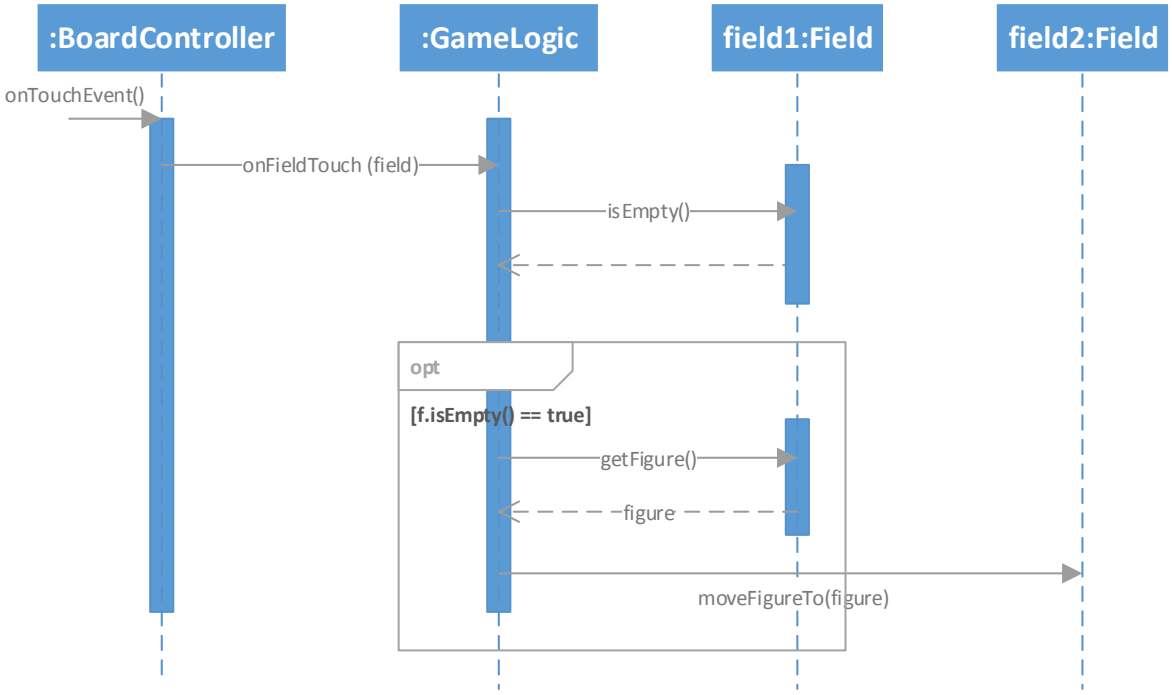
Príloha E Obsah CD

Príloha A



Príloha A Návrh diagramu tried zobrazujúci triedy, ktoré sú relevantné vzhľahom k systému zásuvných modulov.

Príloha B



Príloha B Diagram sekvencie. Ukážka návrhu časti komunikácie pri výbere políčka a následného presunu figúrky z políčka *field1* na políčko *field2*. Z pohľadu natívneho kódu.

Príloha C

Inštalácia

1. Nainštalujeme samostatné Android SDK podľa: https://developer.android.com/sdk/installing/index.html
2. Nainštalujeme vývojové prostredie Eclipse a ADT plugin podľa návodu na: https://developer.android.com/sdk/installing/installing-adt.html
3. Nainštalujeme Android NDK napríklad podľa postupu: https://developer.android.com/tools/sdk/ndk/index.html#Installing
4. V krokoch 1 až 3 sme si pripravili potrebné vývojové nástroje pre Android. Teraz si pripravíme knižnicu na tvorbu doskových hier a jednu doskovú hru. V IDE Eclipse importujeme dva projekty – projekt knižnice *BoardEngine* a projekt hry *BoardGame*: *File* → *Import* → *Existing Projects into Workspace* → *Select archive file* → vyberte súbor `cd://sources/projects.zip`, a následne z ponuky vyberte oba projekty.
5. Preložíme natívnu časť knižnice *BoardEngine* pomocou príkazov v príkazovom riadku, kde `<board_engine>` je cesta k zložke projektu *BoardEngine* a `<ndk>` je cesta k NDK nainštalovanom v kroku 3. Príkaz `ndk-build` vykoná preklad:

```
$ cd <board_engine>
$ <ndk>/ndk-build
```

6. Knižnica *BoardEngine*, vo forme Eclipse projektu, je pripravená k použitiu. Teraz pripojíme knižnicu *BoardEngine* ako závislosť do projektu *BoardGame*:
   a. Označíme si projekt *BoardGame* v Eclipse,
   b. Vykonáme: *Project* → *properties* → *Android* → *Library* → *Add* → *BoardEngine*
7. Hra je pripravená na spustenie na Emulátore alebo reálnom Android zariadení.

Príloha C

Návod na použite

Projekt, ktorý bude používať BoardEngine knižnicu môže byť vytvorený pomocou nasledujúcich krokov alebo je možné použiť kostru projektu s názvom *Skeleton* v archíve `cd://sources/projects.zip`.

1. Predpokladáme, že bola vykonaná inštalácia knižnice BoardEngine.
2. V prostredí Eclipse vytvoríme nový Android projekt:
   a. *File → New → Other ... → Android → Android Application Project*
   b. Zobrazí sa dialógové okno. Zadáme požadované údaje a nastavíme *Minimum Required SDK* na API 16 alebo vyššie. Pokračujeme v procese vytvárania. Na obrazovke *Create Activity* zvolíme zo zoznamu *Blank Activity*.
   c. Dokončíme vytváranie novej aplikácie.
3. Pridáme knižnicu *BoardEngine* rovnako, ako pri inštalácii v kroku 6.
4. Obsah súboru `/res/layout/activity_main.xml` nahradíme napríklad kódom:

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

5. Pripojíme `BoardFragment` do hlavnej aktivity. Konštanta `BOARD_CONFIG_FILE` obsahuje názov konfiguračného súboru , ktorý vytvoríme v nasledujúcom kroku. Triedu hlavnej Aktivity v zložke `/src` nahradíme kódom:

```
public class MainActivity extends Activity {
    private static final String BOARD_CONFIG_FILE = "board_config.xml";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        if (savedInstanceState == null) {
            // vlozenie board fragmentu do aktivity
            getFragmentManager()
                .beginTransaction()
                .replace(R.id.container,
                    BoardFragment.newInstance(BOARD_CONFIG_FILE))
                .commit();
        }
    }
}
```

6. Pripravíme kostru konfiguračného súboru `board_config.xml`, ktorý umiestnime do `/assets`. V konfiguračnom súbore sme nastavili nasledujúce parametre: Herná doska

```xml
<?xml version="1.0" encoding="UTF-8"?>
<board xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="board.xsd">
    <gameLogic>logic.js</gameLogic>
    <materialName>board_material</materialName>
    <layout>
        <squared>
            <width>4</width>
            <depth>4</depth>
        </squared>
    </layout>
    <geometry>
        <source path="board.obj"/>
    </geometry>
   <materials>
         <solidMaterial name="board_material">
            <glossines>0.2</glossines>
            <image path="board.png"/>
        </solidMaterial>
        <solidMaterial name="figure_material_normal">
            <glossines>0.5</glossines>
            <image path="normal.png"/>
        </solidMaterial>
       <solidMaterial name="figure_material_selected">
            <glossines>0.5</glossines>
            <image path="selected.png"/>
        </solidMaterial>
    </materials>
    <figures>
        <figure name="figure">
            <geometry>
                <source path="figure.obj"/>
            </geometry>
            <states>
                <state name="normal">
                    <materialName>figure_material_normal</materialName>
                </state>
                <state name="selected">
                    <materialName>figure_material_selected</materialName>
                </state>
            </states>
        </figure>
    </figures>
   <imageRectangles>
        <rectangle name="logo">
            <image path="logo.png"/>
            <zIndex>5</zIndex>
        </rectangle>
  </imageRectangles>
</board>
```

má rozmery 4x4 políčka, je definovaná geometriou v súbore `board.obj` a je na ňu aplikovaný material *board_material*. Herná logika sa nachádza v súbore `logic.js`. Hra používa tri materiály – jeden pre dosku a dva pre stavy figúrky. Figúrka môže byť v dvoch stavoch a je definovaná geometriou v súbore `figure.obj`. Nakoniec je pripojený popis 2D GUI obrázku s názvom *logo*.

a. (Nepovinne) Konfiguračný súbor môžeme validovať voči schéme `board.xsd`, ktorá by mala byť umiestnená v rovnakom priečinku ako konfiguračný súbor. Schéma sa nachádza v priečinku `cd://others`.

7. Vytvoríme vstupný súbor hernej logiky `logic.js` v zložke `/assets`:

```
function onFieldTouch(x, y) {};
function onEmptyTouch() {};

function onStart() {
    var f = new board.Figure("figure", "normal");
    board.placeOnField(f, 1, 1);

    var f = new board.Figure("figure", "selected");
    board.placeOnField(f, 2, 2);

    var img = new gui.ImageRectangle("logo");
    img.setX(0.25); img.setY(-0.01); img.setWidth(0.5);
};

function onStop() {};
```

8. Doplníme pomocné súbory, ktoré sú vyžadované v konfiguračnom súbore:
    a. `board.obj` – geometria hernej dosky, by mala mať parametre, ako sú popísané v sekcií Návrh konfigurácie hernej dosky. Modely exportované z CAD programov by mali obsahovať normály, UV súradnice a mali by pozostávať iba z trojuholníkov. Export z programu Blender môže prebiehať nasledovne:
        i. Vyberieme exportovaný objekt.
        ii. File →*Export* → *Wavefront (.obj)*
        iii. Označíme nasledujúce možnosti*: Selection Only, Apply Modifiers, Include Edges, Write Normals, Include UVs, Triangulate Faces* a osu do hora – *Up* nastavíme na Z.
    b. `figure.obj` – geometria figúrky sa vytvorí podobne ako geometria hernej dosky.
    c. `board.png` – textúra hernej dosky.
    d. `normal.png` – textúra figúrky v stave *normal*.
    e. `selected.png` – textúra figúrky v stave *selected*.
    f. `logo.png` – obrázok loga.
9. Aplikáciu je pripravená, môžeme ju spustiť. V prípade neúspechu sledujeme *LogCat*, kde sú vypísané chyby, ako napríklad chýbajúce pomocné súbory.

Príloha E

Obsah CD

/assets – pomocné súbory na tvorbu hry
- **/geometry** – .obj súbory a zdrojový blender súbor
- **/textures** – textúry použiteľné na geometriu

/licence – obsahuje súbor s licenciami použitého zdrojového kódu

/others
- **board.xsd** – XML schéma konfiguračného súboru

/sources – Eclipse projekty, ktoré obsahujú zdrojové kódy
- **/BoardEngine** – framework na tvorbu doskových hier
- **/BoardGame** – hra Fox & Geese
- **/Skeleton** – kostra pre vytvorenie hry
- **projects.zip** – archív určený na import projektov do Eclipse

/game – hra Fox & Geese v spustiteľnej forme – APK súbor

/text – PDF a zdrojový text technickej správy