Jihočeská univerzita
v Českých Budějovicích
Ekonomická fakulta

Katedra aplikované matematiky a informatiky

Studijní program: Systémové inženýrství a informatika

Studijní obor: Ekonomická informatika - navazující

Diplomová práce

Vývoj aplikace v programovacím jazyce Java pro operační systém android a její uvedení na trh

Android application development in Java language and its publishing

Autor diplomové práce: Bc. Jakub Bárta

Vedoucí diplomové práce: doc. Ing. Beránek Ladislav CSc.

České Budějovice 2015/2016

Prohlášení

Prohlašuji, že svoji diplomovou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury.

Prohlašuji, že v souladu s § 47 zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své /diplomové práce, a to - v nezkrácené podobě/v úpravě vzniklé vypuštěním vyznačených částí archivovaných Ekonomickou fakultou - elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů.

V Českých Budějovicích dne: Podpis

Obsah

Abstrakt

Tato diplomová práce se zabývá vývojem herní aplikace pro chytré telefony s operačním systémem Android. Je zde popsána situace na trhu mobilních aplikací a vysvětleny základní principy objektově orientovaného programování, Javy a vývoje pro Android. Práce obsahuje i popis umístění finální aplikace na trh a důležitých rysů vývojářských prostředí Eclipse a Android studia. Následuje metodika a podrobný rozbor vývoje aplikace, ve kterém jsou demonstrovány možnosti práce s výše zmíněnými nástroji.

Obě, teoretická i praktická část, mají za úkol nastínit, v jakých situacích se může vývojář aplikací pro OS Android nacházet a na jaké chyby by si měl dát pozor. Téma hry bylo zvoleno tak, aby byla originální, neobsahovala příliš složité grafické prvky, a aby ji uživatel mohl snadno ovládat. Jedná se o rozšířenou verzi šachů, jejíž pravidla jsou vypsána dále v práci. Jednotlivé ukázky vývoje programu jsou zvoleny tak, aby čtenář získal dostatek informací o tom, jak funguje, a jak se při jeho tvorbě postupovalo.

Klíčová slova

Aplikace, Android, Eclipse, chytrý telefon, Java, programování

Cíl práce

Cílem této diplomové práce je popis a demonstrace všech etap vývoje aplikací pro chytré telefony s operačním systémem Android. Úkolem autora tedy není jen tvorba cílové aplikace, ale i seznámení čtenáře s prací ve zvoleném vývojářském prostředí, poukázání na rozdíly v programování počítačové a mobilní aplikace, demonstrace možností vývoje na konkrétním zdrojovém kódu a nakonec podrobný popis umístění výsledného programu na trh. Jako cílová aplikace byla zvolena tahová hra pro dva inspirovaná klasickou verzí šachů, avšak upravená o dodatečná pravidla.

1 Úvod

Tato diplomová práce se zabývá vývojem herní aplikace pro chytré telefony s operačním systémem Android. Jako programovací jazyk zde bude využita Java s vývojářským prostředím Eclipse, které může být rozšířeno o modul pro vývoj Android aplikací.

Účelem této práce je seznámit čtenáře s objektově orientovaným přístupem programování, podrobně rozebrat vývoj aplikace a hlavně popsat možnosti jejího uvedení na trh mobilních aplikací. Jako téma hry byla zvolena upravená verze šachů, kde má každá figurka specifické vlastnosti a pozměněna jsou i některá základní pravidla z původní klasické verze. Tato aplikace nesoucí název Ultimate chess bude uživatelům nabízet pouze možnost hry hráče proti hráči a to jak na jednom společném telefonu, tak i dvou vzdálených zařízeních pomocí serverového připojení.

První, teoretickou částí této diplomové práce bude odborná literární rešerše seznamující čtenáře s historií a problematikou objektově orientovaného programování, operačního systému Android, Javy, prostředí Eclipse a trhu mobilních aplikací. Po seznámení s těmito tématy následuje metodika a praktická část, ve které bude detailně popsán návrh a vývoj programu, jeho vzhled a postup uvedení na trh. Vývoj aplikace bude zachycen pomocí vývojových diagramů a ukázek hlavních částí zdrojového kódu. Postupně budou rozebrány hlavní části všech tříd či důležitých metod programu.

Aplikace byla vybrána tak, aby na ní byl přiblížen objektově orientovaný přístup v praxi a možnosti rozšíření vývojářského prostředí o dodatečné moduly. U této práce se předpokládá, že je čtenář obeznámen s problematikou programování ve zmíněném jazyce Java, a proto zde bude předložen pouze výpis některých částí zdrojového kódu.

2 Teoretická část - literární rešerše

2.1. Trh

Trh je společenská instituce, založená na směně zboží, kterou zprostředkovávají zpravidla peníze. Na trhu se střetávají kupující a prodávající, kteří chtějí prodat a nakoupit za peníze své výrobky a služby. Tyto výrobky a služby označujeme jako zboží. Charakteristickým znakem zboží je, že bylo vyrobeno pro směnu a směněno. Trh nikdo nevymyslel, nenavrhl ani nerealizoval, vznikal v průběhu tisíciletí vývoje zbožní výroby. Je vnitřně strukturovaný, neustále se vyvíjí a vytváří jemné nástroje koordinace činnosti lidí. Trh tedy rozhodně není samoúčelný, má v ekonomice přesně vymezené funkce, které spočívají v hledání odpovědí na známé základní otázky co, jak a pro koho vyrábět z pohledu nabídky a poptávky. (LIŠKA, 2004)

Tato práce se zabývá jednou z nejnovějších oblastí trhu a to trhem mobilních aplikací.

2.1.1 Situace na trhu mobilních aplikací

Statisíce aplikací pro mobilní zařízení, které se objevily v průběhu pár let, dávají tušit, že přichází nová technologická vlna. Smartphonům stačil jen jeden rok na to, aby jejich prodej v zemích západní Evropy a USA překročil počet prodaných jednoduchých telefonů. Neustále rostoucí poptávka po těchto aplikacích naznačuje, že zatím vidíme jen její začátek.
(http://www.softec.cz/reseni/aplikace/vyvoj-analyza-trhu-mobilnich-aplikaci.html, 2016)

Největšími výrobci mobilních aplikací jsou Spojené státy americké, Japonsko a Jižní Korea. Poměrně daleko za nimi se v produkci nachází Čína, Austrálie, Spojené království, Austrálie, Německo, Kanada, Francie a Rusko. Spojené státy dohromady s Japonskem přitom tvoří více příjmů než všechny zbylé, výše vyjmenované země. Tato data jsou počítána na deseti největších trzích v roce 2013.

V rozborech se nejčastěji zkoumají tzv. freemium aplikace, jejichž stažení je zdarma, ale za některý jejich obsah už se platí. Např. při stažení různých her je možné hrát zdarma až do konce, ale často lze přikoupit různá vylepšení, která rozšíří hráčovi možnosti a zjednoduší mu dosažení cíle. Tímto se zamezí

nelegálnímu pirátství, protože aplikaci si může stáhnout každý zdarma, avšak o to víc lze profitovat na prodeji zmíněných rozšíření nebo za přidání reklam.

2.2 Objektově orientované programování (OOP)

„Aristoteles byl pravděpodobně první, kdo začal s důkladným studiem pojmu typ neboli třída. Hovořil o „třídě ryb a třídě ptáků“. Myšlenka, že vše je pouze objekt, přestože jedinečný, jenž je součástí třídy objektů, které mají společné charakteristické vlastnosti a chování, byla u zrodu prvního objektově orientovaného programovacího jazyka Simula-67. Tento jazyk obsahoval nezbytné klíčové slovo „class“, které uvádělo do programu nový datový typ.“ (ECKEL, 2000) Na objektově orientované programování (OOP) se dnes musí vývojáři dívat jako na myšlenku či filozofii, která řeší nový způsob toku informací mezi různými částmi programu. Nejedná se totiž o žádnou předem nastavenou šablonu pro vývoj programu, ale o novou formu myšlení samotných programátorů, na kterou se dnes zaměřuje většina programovacích jazyků.

Jeden ze základních pohledů na problematiku OOP je snaha o abstraktní řešení a simulaci reality. Jinak řečeno se programátoři snaží prozkoumat věc z pohledu člověka, nikoliv stroje. Je zde kladen důraz na to, aby základní části programu mohly být znovu využity a nemusely se znovu vytvářet. Inspirace této myšlenky pochází z dob průmyslové revoluce, kde se v továrnách začaly vyrábět předem vytvořené součástky a nemusely se dělat jednotlivě pro každý produkt zvlášť. Takto předem připravené části už jen stačilo zasadit do finálního objektu (např. motor do automobilu).

2.2.1 Základní složky OOP

Základní jednotkou OOP je objekt, tedy cokoliv co odpovídá něčemu z reality, například již zmíněný automobil nebo i nějaký druh osoby (student, dělník). Objekty se vyznačují tím, že mají své atributy (vlastnosti) a metody. Atributy jsou data, která vyjadřují vnitřní stav objektu. Jedná se o jednoduché proměnné, které určují nějakou vlastnost (věk, jméno, stav, barva, výkon). Metody na rozdíl od atributů neukazují na to, jaký objekt je, ale vyjadřují nám, co dělá. Jinak řečeno se jedná o schopnosti objektu (vykonej, zjisti, brzdi, vypni televizi). Metody lze ve zdrojovém kódu rozlišit podle jednoduchých závorek, které se za ně musí napsat. Tyto závorky mohou obsahovat různé parametry a

celá metoda může vracet nějakou hodnotu. Názorný příklad je třeba metoda seradCisla(), která pracuje s nějakou řadou čísel a řadí je od nejmenšího po největší, např. věk pracovníků ve společnosti. Z abstraktního pohledu může metoda seradCisla() stroji říkat, „Seřaď mi zaměstnance ve firmě podle věku od nejmladšího po nejstaršího.“. Dalším ze základních prvků OOP je třída neboli Class. Na třídu lze nahlížet jako na určitou šablonu pro vytváření objektů, která nám udává, jaké v nich budou atributy a metody. Objekty tohoto typu vytvořené ve třídě jsou nazývány instancemi třídy a dědí společné atributy a vlastnosti, jejichž data se ale mohou lišit. Říká se tedy, že instance třídy mají stejné rozhraní. Například může existovat objekt „Zaměstnanec“, od kterého dědí objekty Dělník a Manažér se společnými metodami pracuj() a podejHlaseni(), ale s jinými hodnotami atributů věk a plat.

Další ze základních myšlenek OOP je zapouzdření. Jedná se o skrývání některých metod či atributů, tak aby je třída mohla použít pouze zevnitř. Tento přístup se používá hlavně jako prevence před chybami. Jinak řečeno programátor předá pouze některé informace o jeho třídě, například návod na její použití nebo jaké vyžaduje parametry, ale odepře přístup k jejímu vnitřnímu kódu, aby zamezil chybám při jeho změnách.

2.3 Java

2.3.1 Popis a výhody Javy

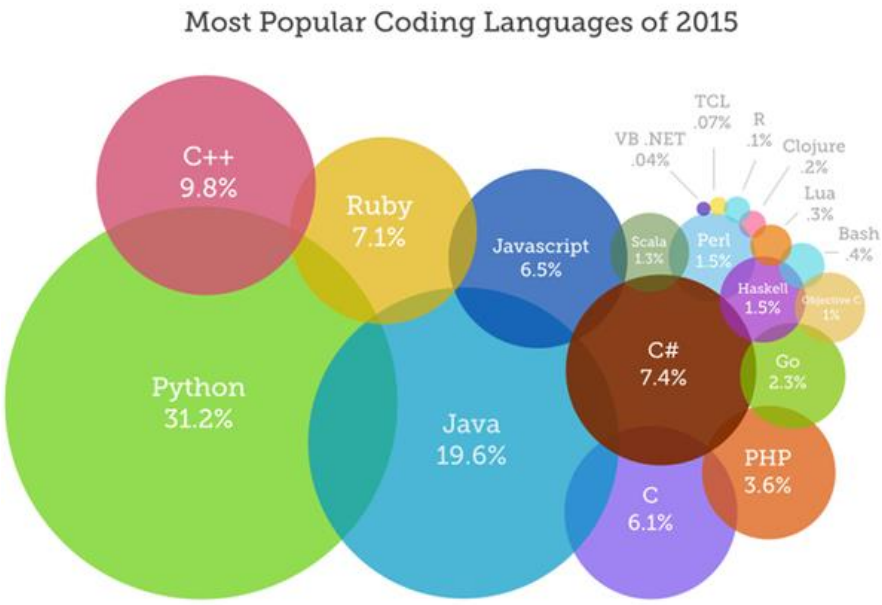
„Java je objektově orientovaný programovací jazyk vyvinutý programátory firmy Sun Microsystems a vycházející z jazyka C++, ke kterému má syntakticky nejblíže. Proti jazyku C++ však Java neobsahuje žádné složité konstrukce způsobující mnoho zbytečných problémů.“ (KISZKA, 2003) Java je tedy vhodná pro programátory, kteří se potřebují plně soustředit na řešení daného problému namísto složitého sestavování kódu, v němž je snadné udělat nějakou chybu. Tento jazyk také obsahuje takzvaný „Garbage collecor[1]“, který se stará o správu paměti a maže nepotřebné objekty, v praxi je toto skvělá pomůcka. Pro její uživatele působí Java velice příjemně a spolehlivě. Další výhodou je, že programy napsané v Javě se dají využít na všech platformách, například na

[1] Garbage collector – „sběrač odpadu“

chytrých telefonech a tabletech, které nám v dnešní době začínají nahrazovat počítače.

Java je dnes tak rozšířená a využívaná, protože se snaží programátorům poskytnout prostředí pro co nejjednodušší psaní kódu, aby mohli veškeré úsilí soustředit na řešení projektu. To všechno Java umožňuje díky své jednoduché syntaxi, kterou lze poměrně snadno pochopit. Vedle již zmíněného Garbage collectoru je výhodou Javy také její přenositelnost a je vhodná pro projekty všech rozsahů. Program napsaný v tomto jazyce totiž funguje u všech operačních systémů. Java je velice vhodná i pro nováčky, ti si mohou snadno vyhledat chyby, kterých se z počátku dopouští většina programátorů. Další výhodou je rozsah jejího rozšíření ve světě. Jinak řečeno se snadno hledá zdroj, ať už osoba nebo fórum, ze kterého si lze vytáhnout potřebné informace k řešenému problému. Např. na internetu lze najít tisíce řešených situací, ze kterých je užitečné se poučit. Znalost Javy se také uvádí v mnoha podmínkách pro získání pozice na trhu práce. Nemalou zásluhu na kvalitě tohoto jazyka mají vývojová prostředí (Netbeans, Eclipse) a široké spektrum přídavných modulů a knihoven, které rozšiřují možnosti a usnadňují práci.

Obrázek 1: Podíl nejpoužívanějších programovacích jazyků v roce 2015



(Devsaran, © 2016)

2.3.2 Historie Javy

„Společnost Sun Microsystems původně vyvinula Javu jako jazyk pro řízení set-top přístrojů pro televize. (Set-top přístroje se podobají přístrojům příjmu kabelové televize, ale nabízejí mnohem větší interakci a více možností než dnešní přijímače.) Společnost tedy chtěla vyvinout operační prostředí pro spotřebitelská zázemí, která podávají informace přes kabelovou televizi.“ (PERRY, 1996) Původní projekt byl započat v roce 1991 pod názvem „Green“ team (zelený tým). Tento menší tým pod vedením Jamese Goslinga navrhl a vytvořil původní verzi Javy. Hlavní myšlenkou tohoto projektu bylo vyvinutí systému řízení elektroniky, který nebude závislý na platformě. Jinými slovy nebude potřebovat žádné konkrétní mikroprocesory či hardwarové součástky a bude pracovat jak na videorekordéru Sony, Applu nebo jiném, ale i na bezdrátovém telefonu a dalších zařízeních. Vývoj tohoto softwaru pro řízení elektroniky byl velmi náročný. Ukázalo se, že musí být nejen adaptivní, ale i zpětně kompatibilní. Toto je dáno tím, že některé zařízení mají mnohem delší životnost než počítače (např. 20 let stará sušička), ale software se všeobecně každou chvíli mění a vylepšuje. Od spotřebitelů byla také vyžadována veliká spolehlivost softwaru, protože když se něco porouchá v systému osobního počítače, lze ho opravit, ale jiný přístroj s porouchaným systémem (mikrovlnná trouba) se většinou vyhodí. Projekt Green později přejmenovaný na Oak se poté krátce zaměřil na první herní konzole s CD-ROM, ale hlavní úspěch oslavil s myšlenkou zapojení se do vývoje webových aplikací. V roce 1995 společnost Oak zjistila, že toto jméno si už v minulosti někdo zaregistroval a nechala tento jazyk přejmenovat na „Java“. Někteří lidé tvrdí, že toto slovo je zkratkou pro „Just Another Valueless Akronym“ (jen další bezcenný akronym), to ale společnost popírá. Během roku 1997 byl počet stažení Javy během prvních třech týdnů po vydání vyšší než 230 000 a o rok později už jí užívalo přes dva miliony lidí. (BÁRTA, 2014)

2.3.3 Verze Javy a zvolené vývojové prostředí

Pojem Integrated Development Environment zkratkou IDE[2], se využívá pro vývojové prostředí, ve kterém programátor pracuje na svém zdrojovém kódu. Tento druh softwaru je většinou vyvinut pro jeden programovací jazyk a měl by obsahovat pomůcky ke zrychlení a usnadnění vývoje. Mezi hlavní nástroje patří editor zdrojového kódu, kompilátor a debugger.

U tohoto projektu bylo využito vývojové prostředí Eclipse verze „Luna Service Release 2 (4.4.2)“. Toto IDE pomohlo hlavně svou podporou pro vývojáře Android aplikací. K tomu stačilo stáhnout a nainstalovat příslušný modul Android studia, který nabízí např. šablony pro vývoj nebo v něm lze nasimulovat obrazovku vybraného chytrého telefonu.

2.3.4 Eclipse

Eclipse je integrované vývojové prostředí (IDE), obsahuje základní pracovní prostor pro vývoj aplikací, který je díky plug-in systému možno různě rozšířit a přizpůsobit. Samotné Eclipse je napsáno převážně v Javě a nejvíce se využívá právě pro řešení Java projektů. Díky široké škále zásuvných modulů lze toto IDE použít i k programování v jiných jazycích, např. v C, C++, JavaScript, PHP, Python, R a dalších. Vydáno v souladu s podmínkami Eclipse Public License je Eclipse SDK je zdarma a uživatelé mohou přispívat vlastními moduly, jedná se tedy o open-source software.

2.4 Android

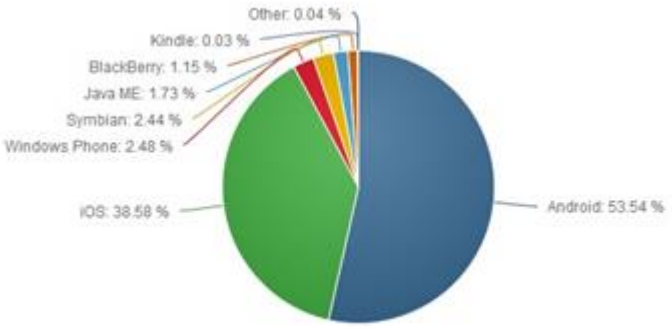
Android je mobilní operační systém založený na jádře Linuxu. Jedná se o open source software, který byl vyvinut pro snadný a nenákladný rozvoj mobilních technologií. Je navržen tak, aby bral v potaz parametry zařízení, na kterém je nainstalován (velikost obrazovky, kapacita baterie).

Android je nejrozšířenějším operačním systémem na trhu s chytrými telefony. V srpnu roku 2014 byl nainstalován na 84,7 % zařízení. Svou vedoucí pozici si drží i dnes na úkor konkurenčních operačních systémů pro mobilní zařízení (iOS, Windows Phone, BlackBerry OS a další). Rychlost růstu Androidu dokazuje i frekvence s jakou vychází jeho nové verze. Tento systém však

[2] IDE – Integrated Development Environment

nevyužívají pouze chytré telefony, ale i tablety, televize, set-top boxy, hodinky a Google TV. Do budoucna je v plánu zařadit Android i do ovládání automobilů.

Obrázek 2: Podíl operačních systémů pro chytré telefony v roce 2015

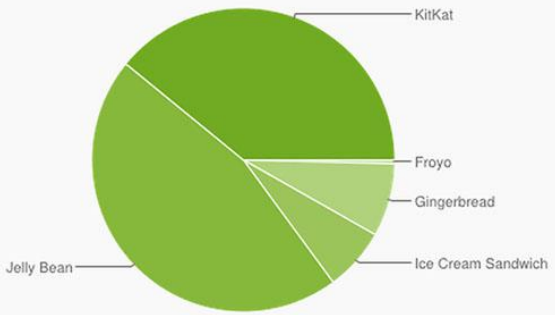


(Windows 8 Core, © 2012-2015)

Na obrázku výše lze vidět dominující podíl zařízení, které v roce 2015 používaly OS Android. Nejčastěji šlo o verze KitKat a Jelly Bea. V současnosti se zvedá tržní podíl s iOS operačním systémem pro Apple.

Obrázek 3: Podíl používaných verzí Androidu v roce 2015

Version	Codename	API	Distribution
2.2	Froyo	8	0.4%
2.3.3 - 2.3.7	Gingerbread	10	7.8%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	6.7%
4.1.x	Jelly Bean	16	19.2%
4.2.x		17	20.3%
4.3		18	6.5%
4.4	KitKat	19	39.1%



(DRD Life Inc., © 2008-2016)

2.4.1 Historie Androidu

Společnost Android Inc. odkoupil v roce 2005 Google a udělal z ní svou dceřinou firmu. Po odkoupení společnosti byl do vedení dosazen Andy Rubin a byl to právě on, kdo započal vývoj systému. V roce 2007 Google získal několik patentů týkajících se mobilních zařízení. Tím se také světem začaly šířit první spekulace o možném vstupu Googlu na trh chytrých mobilních telefonů a 5. listopadu roku 2007 nastává jeden z nejdůležitějších kroků v historii společnosti. V tento den byla vytvořena tzv. „Open Handset Alliance“. Jedná se o seskupení výrobců mobilních telefonů. Cílem tohoto uskupení bylo vyvinout standard pro mobilní zařízení. Dnešními členy konsorcia jsou například: Google, Intel, Samsung, HTC, LG, NVIDIA, Qualcomm, Sony, Dell, Telefónica, T-Mobile, eBay a asi dalších 70 společností z celého světa. Ještě v ten samý den Google oznámil první verzi mobilního operačního systému Android. Trvalo téměř celý rok, než byl vydán první telefon s Androidem. Jeho výrobcem byla společnost HTC a ve Spojených Státech byl uveden na trh v říjnu roku 2008. Jednalo se o HTC Dream, v některých částech světa prodávaný pod názvem T-Mobile G1. Na český trh se dostal počátkem roku 2009.
(http://www.svetandroida.cz/kratke-ohlednuti-za-historii-androidu-201305, 2016)

2.4.2 Android studio

Android studio je oficiální IDE, které se velmi často využívá pro vývoj android aplikací. Jedná se o společný produkt společností Google a JetBrains, je k dispozici veřejně, zcela zdarma a poskytuje vývojářům modul k rozšíření ostatních IDE. Android studio je založeno na bázi IntelliJ IDEA, o kterém se píše jako o jednom z nejinteligentnějších vývojářských prostředí pro Javu. Navíc obsahuje buildovací nástroj Graddle, možnost generování apk souborů nebo různé emulátory, na kterých lze simulovat vyvíjenou aplikaci. V neposlední řadě má k dispozici layout editor, s jehož pomocí si lze jednoduše vybrat prvky uživatelského prostředí, jako jsou např. tlačítka nebo textová plocha a přetáhnout je přímo do své aplikace bez nutnosti psaní kódu. U tohoto projektu je využito jak Android studio, tak i jeho modul pro Eclipse.

2.4.3 Rozdíl mezi vývojem desktopové a mobilní aplikace

Pro tuto práci a usnadnění vývoje byla výsledná aplikace nejprve vyvinuta pouze pro počítače. Z této verze se využilo zhruba 70% kódu, o který se autor mohl opřít při přechodu na Android. Otázkou tedy zůstává, v čem se liší těch zbylých 30%. Obecně lze říci, že hlavní rozdíl je v uživatelském prostředí a v hardwarových prvcích telefonu a počítače, které jsou značně odlišné.

Jeden z hlavních problémů vývoje mobilních aplikací je velikost obrazovky. Je potřeba brát v potaz, že telefony mají poměrně malé displeje a přestože mnoho z nich oplývá HD rozlišením, tak jejich velikost dosahuje pouze od tří do šesti palců. Nejčastěji užívaná velikost obrazovky se pohybuje mezi 4 až 4,7 palců.

Dále je nutno vyřešit uživatelsky pohodlné zadávání vstupů a způsob zobrazování výstupů. Např. tlačítko s pevným rozlišením v pixelech může být snadno použitelné na čtyřpalcovém zařízení, ale na tabletu by byla jeho funkčnost mizivá. Proto je třeba všechny komponenty upravit tak, aby se správně zobrazovaly na všech velikostech obrazovek. Je také třeba brát v potaz, že někteří uživatelé mají velké prsty, a proto by tlačítka měla být dostatečně velká. Vstup se ale nezadává pouze pomocí tlačítek, ale i jinými způsoby např. psaním textu. Při zadávání textu překrývá virtuální klávesnice část displeje, někdy i celý a to může narušit pohodlné užívání některých aplikací. Proto je potřeba zobrazování klávesnice upravit dle potřeb nebo úplně vynechat a vstup zadávat jinak. V konkrétní situaci lze vyměnit např. zadávání čísla za dvě tlačítka plus a mínus. Tento postup se používá u kalendářů či budíků a zamezí se tím zobrazení klávesnice.

Při vývoji uživatelského rozhraní je důležité promyslet, v jaké poloze se bude zařízení držet. Podle toho lze správně rozvrhnout umístění tlačítek a ostatních komponent. Na zobrazenou část aplikace tedy nelze pohlížet stejně jako u počítače, ale je nutno si uvědomit, že se jedná o interaktivní rozhraní, kde probíhá komunikace mezi dotykem člověka a obrazovkou. Z tohoto důvodu by se v hlavních menu mělo první tlačítko nacházet zhruba na středu displeje, protože tam naše palce dosáhnou nejsnáze. Vývojář si může i rozmyslet, jestli má smysl povolit systému automatické přetáčení obrazovky. Tato funkce totiž nemusí být vždy pouze přínosem.

U hardwarových požadavků by se měl klást důraz hlavně na vytížení procesoru a správně uvolňovanou alokační paměť. Ne každý vlastní zařízení s osmi jádry o taktu 2,4 GHz a je tedy třeba aplikaci upravit tak, aby nezamrzávala a neomezovala běžný chod průměrného chytrého telefonu. Nejčastější chybou nováčků je použití velkého množství animací, které výkon telefonu nestíhá načítat. Dalším hardwarovým problémem je úložiště. Android automaticky instaluje stažené aplikace do interní paměti, která se velmi brzo zahltí. Existují i postupy, jak tyto aplikace instalovat rovnou do paměti externí, ty však většinou vyžadují tzv. root a ztrátu záruky. Obecně platí pravidlo, čím je aplikace menší, tím je uživatelsky pohodlnější.

Hardwarové doplňky chytrých telefonů, mají i své výhody. Nejvýraznější je asi možnost přenosného internetu, dále může být k aplikacím využita kamera, mikrofon, GPS či Gyroskop, který určuje, jestli se zařízení nachází ve svislé nebo vodorovné poloze.

2.4.4 Uvedení na trh

Aby si výsledný produkt mohlo stáhnout co nejvíce lidí, je potřeba ho umístit na trh. Online obchod s Androidovými aplikacemi se nazývá Google play. Předtím než může poslat svou aplikaci do oběhu, musí vydavatel udělat několik kroků. Nejdříve ze všeho je nutné převést hotový projekt do formátu .apk, který u Androidu slouží podobně jako .exe soubory u Windows. O tuto konverzi se postará používané IDE. Dalším krokem je registrace vydavatelského účtu u Google play za poplatek 25$. Po přihlášení ke svému účtu je možno nahrát zvolený apk soubor a nastavit podrobnosti o distribuci aplikace. Povinným nastavením je výchozí jazyk, název, obrázek loga, obrázek aplikace, cena a regiony distribuce. Volitelným nastavením je např. popis aplikace, snímek obrazovky nebo nastavení cen pro každý region zvlášť. Po splnění všech mandatorních požadavků je možno aplikaci nahrát na trh, kde ji po osmi až deseti hodinách bude možno stáhnout. Detailnější popis s obrázky postupu bude vypracován v praktické části.

2.5 Programování v Android studiu

2.5.1 Aktivity

Aktivitu lze popsat jako základní komponentu vykreslování grafiky určité části projektu. Například po zapnutí aplikace se může zobrazit nějaká prvotní animace, ať už se jedná o logo či reklamu, která uživatele uvítá do programu. Právě tato animace je první aktivitou. Jako druhá se může zobrazit hlavní menu, které po kliknutí na tlačítko přesměruje na třetí aktivitu a takto to pokračuje dál. Některé aktivity, např. u volání jsou již v telefonu definované a lze k nim v projektu přistupovat.

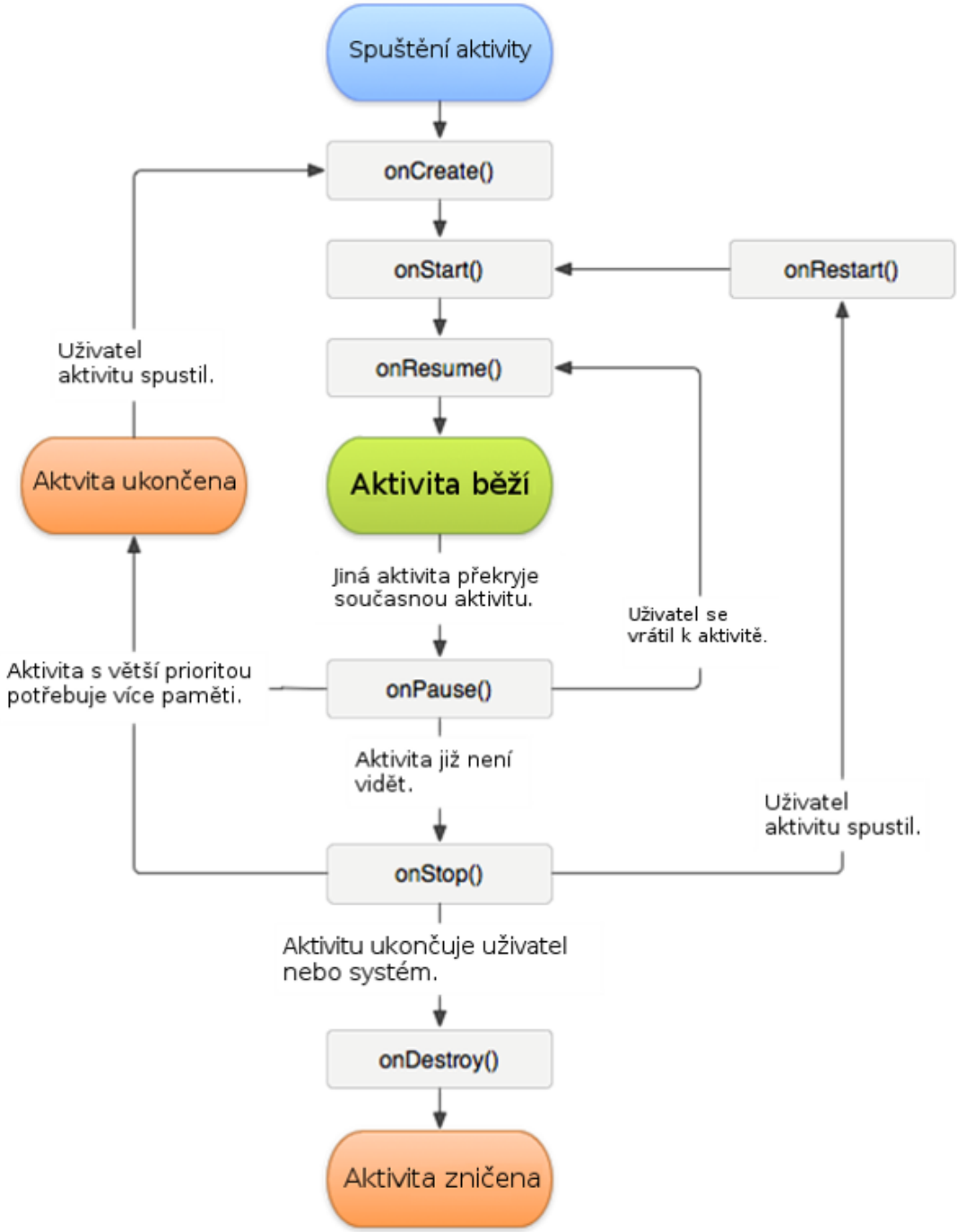
2.5.2 Životní cyklus aktivity

Jelikož je Android operační systém určený převážně pro chytré telefony, zodpovídá za své aktivity a za vše, co s nimi souvisí. Jinými slovy má určité primární nastavení, jak se má v některých situacích zachovat. Dobrým příkladem je přerušení běžné herní aplikace, když v tu samou chvíli uživateli někdo zavolá. Android v tu chvíli tedy určil prioritu pro vyřízení hovoru. Aplikaci může přerušit nebo pozastavit i sám hráč, když se např. vrátí do hlavního menu. Nelze tedy dopředu přesně říci, jak dlouho aktivita poběží, a proto se v různých situacích používají metody, který daný problém vyřeší.

Stavy aktivit:

1. **Běží** – Po úspěšném spuštění běží aktivita na popředí obrazovky, uživatel ji tedy může vidět.
2. **Pozastavená** – Aktivita je stále vidět, ale může nastat situace, kdy ji překryje jiná aplikace, např. volání, SMS nebo zpráva o vybité baterii. Uživatel má v tu chvíli znemožněn přístup k aplikaci a nemůže s ní nijak pracovat.
3. **Zastavená** – Uživatel už aktivitu nevidí, nemá k ní přístup a nemůže s ní pracovat. Její objekt však nebyl úplně zničen a lze se k ní ještě vrátit.
4. **Ukončená** – K takové aktivitě se už nelze vrátit, je zničena.

Obrázek 4: Životní cyklus aktivity



(itnetwork.cz, © 2016)

Metody životního cyklu:

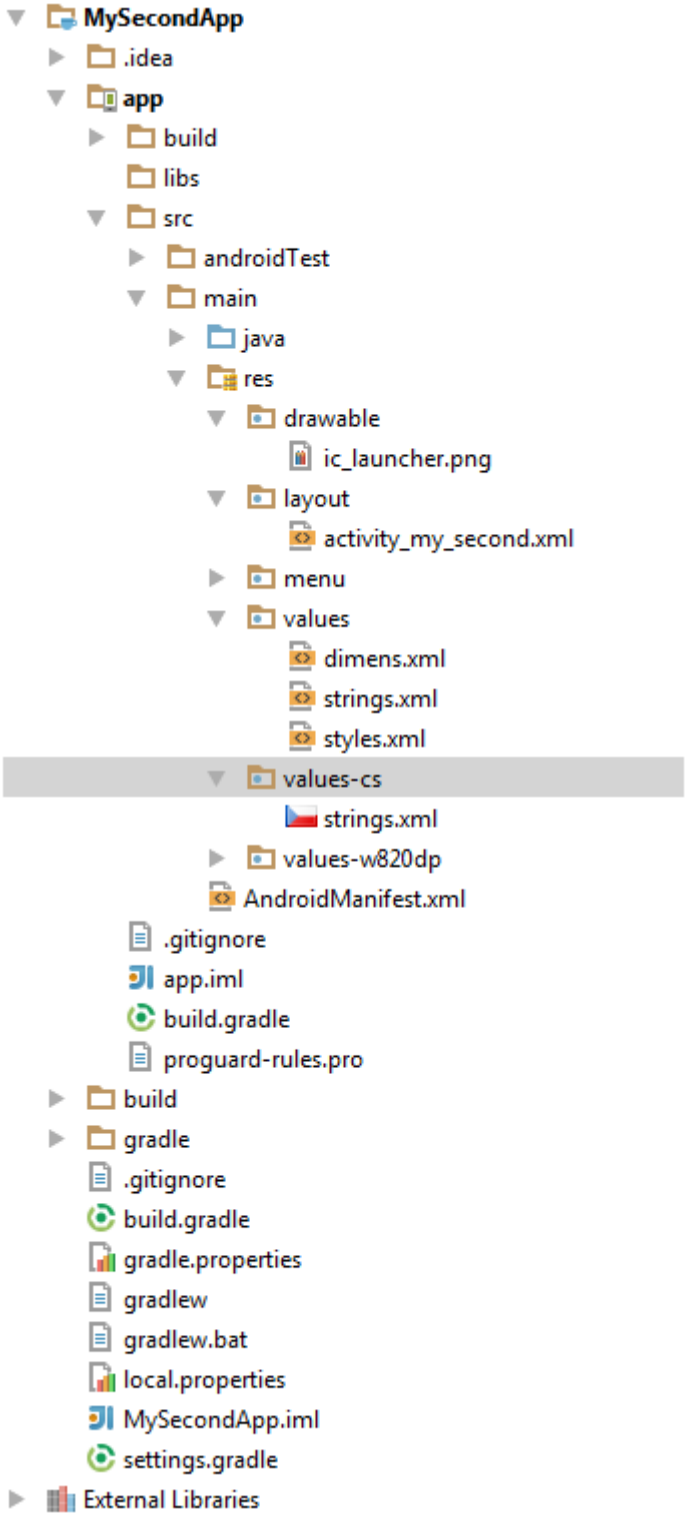
- **onCreate()**, během spouštění nějaké aktivity má Android na starosti základní věci jako vytváření objektu nebo spouštění celého procesu. Hned na to se zavolá metoda **onCreate()**, která má za úkol nadefinovat veškeré nezbytnosti ke spuštění aktivity, např. jaké grafické rozhraní se má zavolat a kam se má zobrazit.
- **onStart()** se při prvním spuštění volá po metodě **onCreate()** nebo pokud byla aktivována jinou událostí, např. příchozí SMS nebo po zobrazení dialogu o nabité baterii. Tato metoda nemůže dostat uživatelský vstup.
- **onResume()** je volána těšně před událostí, která přenese aktivitu do popředí (po zrušení pauzy ve hře, restartování) a může dostat uživatelský vstup.
- **onPause()**, tato metoda se zavolá při přechodu aktivity na pozadí. Dává systému oprávnění aktivitu násilně ukončit.
- **onStop()** není pro uživatele nijak viditelná a má za úkol zastavení aktivity.
- **onDestroy()** se zavolá před samotným zrušením aktivity.
- **onRestart()**, pokud byla zavolána metoda **onStop()** a aktivita byla restartována, tak je následně zavolána **onRestart()**.

Všechny tyto metody v projektu nemusí být využity, někdy jich jednoduše není potřeba. Jedinou výjimkou je **onCreate()**, která je nutná u každé aktivity.

2.5.3 Struktura souborů v projektu

Při programování v Android studiu je nutno porozumět struktuře jeho souborů a složek. Každá taková složka má svůj účel a patří do ní jiná část kódu. Pro ilustraci je níže zobrazená struktura těchto složek.

Obrázek 5: Struktura souborů v Android studiu



App

Ve složce **app** se nachází další tři podsložky:

- **build**, do této části vývojář nemusí nic psát, slouží totiž pouze jako adresář pro ukládání předkompilovaných kusů kódu, při každém spuštění aplikace.
- **libs**, jak už napovídá zkratka pro knihovny, z anglického libraries, slouží tato sekce pro umístění nových rozšíření a přidávání různých funkčností projektu.
- **src**, zde se nachází důležitý **main**.

Main se dále větví na **java** a **res**:

- **java**, sem se píše Java kód přiřazený aktivitám.
- **res**, pod zkratkou pro resources se nachází úložiště pro obrázky použité v aplikaci. Tyto obrázky se podle kvality rozlišují a umisťují do jednotlivých adresářů. Pro nejméně kvalitní obrázek se použije adresář drawable-hdpi a ty nejkvalitnější full HD obrázky jsou umístěny v drawable-xxhdpi. Tyto adresáře jsou ve složce **res** celkem čtyři.
    - drawable-hdpi
    - drawable-mdpi
    - drawable-xhdpi
    - drawable-xxhdpi

Pokud se jedná o jednodušší projekt, lze všechny tyto adresáře nahradit souhrnným adresářem drawable a umístit všechny obrázky do něj.

Values

Zde se nachází proměnné celé aplikace, definující např. barvy, texty atd. V této složce se názorně projevuje styl objektově orientovaného programování. Časově je mnohem úspornější odsud stokrát zavolat nějakou neznámou, než ji stokrát vytvářet jednotlivě. Při změně hodnoty této proměnné ji stačí přepsat pouze zde a výsledek se projeví na celé aplikaci.

Layout

Do této složky se ukládají soubory ve formátu xml, ve kterých je definováno kde, jak a co se má vykreslovat. Jinými slovy je zde určeno, jak uživatel uvidí obrazovku.

Manifest

Toto je asi nejdůležitější složka struktury. Je nutné, aby se tento klíčový soubor AndroidManifest.xml nacházel v kořenovém adresáři každé aplikace. Obsahuje totiž informace, které se musí Androidu předat už před samotným spuštěním kódu. Jsou zde definována např. práva aplikace, její ikona a nachází se zde i některé aktivity.

2.5.4 Zisk plynoucí z aplikací

Co motivuje programátory, aby vyvíjeli nové aplikace? Pro někoho je dostatečnou motivací pouze dobrý pocit, že si jeho výtvor stáhlo mnoho lidí, ale většině z nich jde hlavně o zisk. Ten lze touto formou získat hned několika způsoby. Prvním je klasický prodej své aplikace za předem stanovenou cenu. Už s tím má ale velká část uživatelů problém a nejsou zvyklí za tyto služby platit. Proto se na trhu vyskytují aplikace, které jsou ke stažení zdarma. Takové aplikace mohou přinášet profit např. přidáním reklam, jejichž majitelé vyplácí určitou drobnou peněžní částku, pokud na ně uživatel klikne. Další metodou je nabídnout aplikaci zdarma, ale přidat do ní nějaké zpoplatněné prvky, jako jsou různá rozšíření, usnadnění nebo i zrušení zmíněných reklam.

Při rozhodování jaký marketingový tah zvolit se vývojář pohybuje na velmi tenkém ledě. Pouhých 10 Kč totiž může odradit zákazníky od stažení aplikace, stejně tak i příliš rušivé reklamy nebo obchodní strategie, jako je např. „Pay to Win“, kde jsou hráči, kteří si připlatili výrazně ve výhodě. Na Google play se obecně nevyplácí stanovit si vysokou cenu za stažení. Tomu je trh s aplikacemi pro Apple mnohem vstřícnější, zákazníci jsou tam totiž zvyklí si za služby připlatit.

3 Metodika

I. Literární průzkum

Pro tuto diplomovou práci je potřeba nejprve udělat literární průzkum a získat data nejen z knižních zdrojů, ale hlavně internetových. Jelikož se jedná o velice moderní téma, jsou tyto zdroje prioritou. Potřebné informace se týkají především objektového programování v Javě a potřebných znalostí pro vývoj Android aplikací. Až po vypracování této části bude možné začít s tvorbou finální aplikace.

II. Analýza

První fází tvorby aplikace je nutné určit její účel. Výsledný program má za úkol nejen demonstrovat možnosti vývoje, ale měl by být navržen tak, aby mohl uspět na trhu s mobilními aplikacemi. Toho se autor pokusí dosáhnout odhadem situace na trhu, originalitou a zábavností hry a také několika návrhy pro zvýšení ziskovosti aplikace.

III. Hrubý návrh aplikace

Při návrhu aplikace je nejprve nutno ujasnit si funkce jednotlivých základních komponent. K tomu slouží základní obraz projektu, který bude obsahovat pouze prvky jako je program, uživatel nebo zařízení a jejich vztahy.

IV. Diagram tříd

Diagram tříd vývojáři poslouží nejen jako pomůcka pro lepší orientaci při tvorbě aplikace, ale i jako úložiště jeho nápadů a myšlenek. Budou zde graficky znázorněny jednotlivé komponenty aplikace, jako jsou třídy a metody a jejich následná komunikace.

V. Výběr programovacího jazyka

U tohoto projektu byl vybrán programovací jazyk Java. K výběru pomohl literární průzkum, ze kterého vhodnost Javy vyplynula díky její rozšířenosti a možnosti adaptovat se na různé platformy.

VI. Návrh uživatelského prostředí

Dalším krokem je návrh prostředí, ve kterém se bude uživatel aplikace orientovat. K tomu je dobrou pomůckou návrhová sekce Android studia, ve které si lze udělat rychlou a poměrně přesnou představu, jak bude uživatelské prostředí vypadat.

VII. Implementace kódu

Zdrojový kód bude podle diagramu rozdělen do několika tříd, které budou mít za úkol vytvořit jednotlivé části hry, jako je například šachovnice nebo figurky a jejich vlastnosti. Další třídy se budou starat o chod aplikace, interakci s hráčem a o zobrazování uživatelského prostředí.

VIII. Uvedení na trh

Aplikace bude po jejím dokončení umístěna na trh mobilních aplikací, podle postupů zjištěných z literárního průzkumu.

4 Praktická část - aplikace

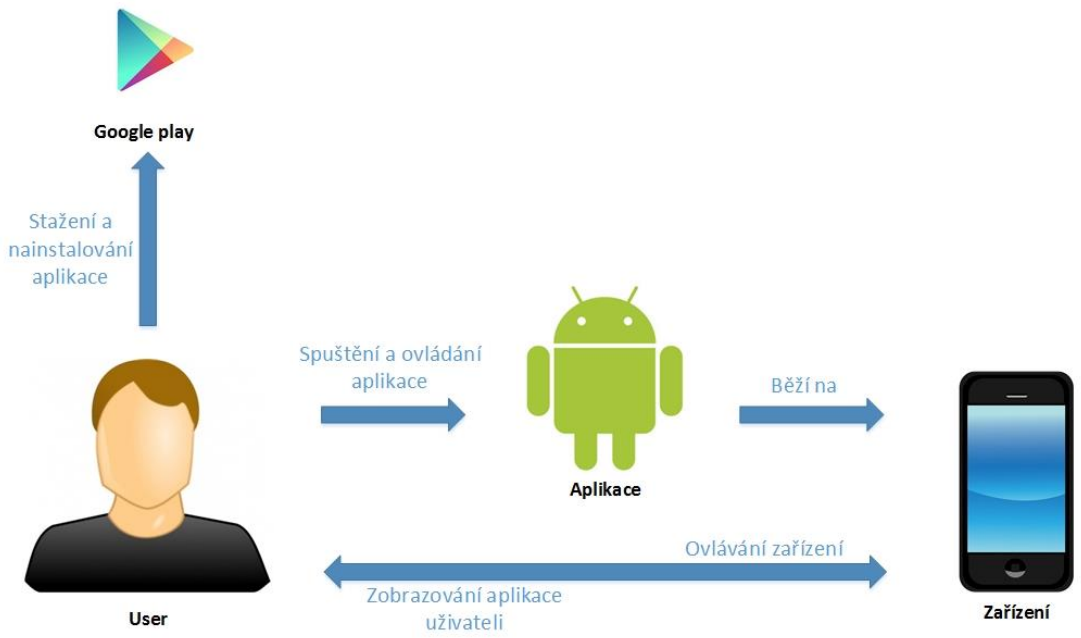
4.1 Popis a návrh aplikace

Jak již bylo zmíněno, cílem této práce je vytvořit herní aplikaci pro OS Android. Hra se sice inspiruje původní verzí klasických šachů, ale její pravidla jsou značně pozměněna, proto dostala název Ultimate chess. Cílem je vytvořit pouze herní mód hráče proti hráči.

4.1.1 Hrubý návrh základních komponent

Před psaním kódu, je důležité uvědomit si co s čím a jak bude komunikovat. K tomu je dobré si vytvořit nějaký grafický návrh, který pomůže programátorovi v lepší orientaci během vývoje.

Obrázek 6: Hrubý návrh základních komponent aplikace



Obrázek výše popisuje vztah uživatele, aplikace a zařízení, na kterém poběží. Hotová hra bude umístěna na trh pomocí stránek Google play, odkud si ji uživatel bude moci stáhnout a nainstalovat do svého zařízení. Tím role uživatele samozřejmě nekončí, musí ještě aplikaci zapnout a ovládat nejen ji, ale i zařízení, na kterém ji spustí. Toto zařízení, v našem případě chytrý telefon, mu pak graficky zobrazí uživatelské rozhraní.

4.1.2 Návrh uživatelského prostředí

Uživatelské prostředí, je velice důležitou částí návrhu, kterou je potřeba dobře promyslet. Uživatel totiž hodnotí aplikaci většinou podle toho, jak vypadá a jak snadno a pohodlně se ovládá. Zde se prostředí skládá ze dvou hlavních sekcí, ve kterých se hráč bude pohybovat a to z hlavního menu a herní obrazovky.

Hlavní menu

Obrazovka, která hráče přivítá po zapnutí aplikace, se skládá z grafického podkladu v pozadí, ve kterém je založeno celkem pět tlačítek:

- **New game**, po kliknutí na toto tlačítko se spustí herní obrazovka.
- **Settings**, zde se po kliknutí zobrazí protokol pro nastavení hry.
- **Shopping**, tato sekce by měla sloužit pro nákupy herních balíčků.
- **About**, zobrazuje informace o samotné aplikaci a jejím tvůrci
- **Quit**, po kliknutí vrátí uživatele na domovskou obrazovku v telefonu.

Herní obrazovka

Tato obrazovka se zobrazí po kliknutí na tlačítko Nová hra v hlavním menu. Důležité jsou zde tři části:

- **Pravý panel**, zde se po kliknutí na figurku zobrazí nápověda a informace, které se k dané figurce vztahují včetně obrázku s možnostmi jejího pohybu. Dále je zde vypsáno, který hráč je na tahu. Tento panel slouží pro lepší orientaci ve hře.
- **Šachovnice,** je nastavena tak, aby se přizpůsobila displeji a její velikost byla od dolní hranice až po horní. Je to nejdůležitější herní plocha, jsou na ní zobrazeny všechny figurky a celá reaguje na dotyk uživatele.
- **Pravý panel,** zde je vyhrazen prostor pro reklamu a pro čtyři tlačítka. Těmi jsou vracení tahu zpět a zase dopředu, otáčení šachovnice a vrácení zpět do hlavního menu.

4.1.3 Návrh tříd a objektů

Při programování se nejlépe postupuje rozebíráním problému na menší a menší části. Další krokem v návrhu je proto vytvoření diagramu tříd, který už popisuje komunikaci jednotlivých tříd, metod a ostatních komponent projektu. Pokud se tato část provede správně, neměl by už programátor strávit příliš času nad změnami v návrhu a řešením problémů. Ušetřený čas pak může využít na psaní a optimalizaci kódu. Následující tabulka slouží pouze jako výpis tříd a rozhraní a jejich významu. Samotný diagram tříd je připojen v příloze.

Tabulka 1: Seznam tříd / rozhraní a popis jejich funkce

Název třídy / rozhraní	Popis
Figurka	Udává vlastnosti všech figurek. Určuje jejich možnosti pohybu i útoku. Obsahuje konstruktor pro vytvoření nové figurky.
Bunka	Nastavuje vlastnosti polí šachovnice. Určuje velikost, barvu, polohu políček atd.
Sachovnice	Vytváří samotnou šachovnici složenou z buněk, naplňuje ji figurkami, vykresluje ji.
Menu	Tvorba hlavního menu aplikace
PraveVybranaFigurka	Pomocná třída pro uložení momentálně vybrané figurky
Game	Nastavuje některé funkce a tlačítka v herní obrazovnce
LevyPanel	Nastavuje panel na levé straně herní obrazovky
PravyPanel	Nastavuje panel na pravé straně herní obrazovky
Setting	Řeší ukládání ukládání dat s preferencemi
About	Pomocná třída pro nastavení aktivit
IfaceFigurka	Pomocné rozhraní pro zobrazování pohybu a útoku

4.1.4 Pravidla hry

K pochopení následujících částí je potřeba vypsat si všechna pravidla, kterými se hra řídí. Ultimate chess se odehrává na klasické šachovnici 8 x 8 polí se stejným počátečním rozmístěním figurek, jako při normální šachové partii. Každá z těchto figurek má své specifické jméno, vzhled a vlastnosti. Hráči se střídají po každém tahu. Všechna pravidla jsou vypsána níže.

- **Figurky**
  - Master (král)
  - Blade dancer (královna)
  - Frost tower (levá věž)
  - Fire tower (pravá věž)
  - Wizard (levý kůň)
  - Stomper (pravý kůň)
  - Cannon (levý střelec)
  - Ranger (pravý střelec)
  - Pawn (pěšec)
- **Cíl hry** – zabít soupeřova mastera
- **Životy** – na rozdíl od normálních šachů má zde každá figurka určitý počet životů. Pokud je počet životů roven nule, figurka zemře.
- **Pohyb** – každá figurka má specifický pohyb.
- **Útok** – útok se provádí vždy a pouze po pohybu. Bez provedení pohybu nelze zaútočit. Všechny figurky útokem berou jeden život, pokud není určeno jinak.

Obrázek 7: Vzhled figurek

Vlastnosti jednotlivých figurek:

- **Master**
  - Životy: 3
  - Pohyb: 1 pole diagonálně (uhlopříčně) i axiálně (dopředu, dozadu a do stran)
  - Útok: 1 pole jakýmkoliv směrem, zasaženou figuru okamžitě zabije a stoupne si na její místo.
  - Schopnosti: Master si místo vykonání pohybu může prohodit pozici s jednou ze svých věží, útok se však neprovede.
- **Blade dancer**
  - Životy: 2
  - Pohyb: neomezeně polí diagonálně i axiálně
  - Útok: plošný útok 1 pole kolem sebe
- **Frost tower**
  - Životy: 2
  - Pohyb: 2 pole axiálně
  - Útok: 2 pole axiálně
  - Schopnosti: Zasaženou figuru zamrazí a znemožní její pohyb v příštím kole.
- **Fire tower**
  - Životy: 2
  - Pohyb: 1 pole axiálně
  - Útok: 3 pole axiálně
  - Schopnosti: Zasaženou figuru posune o pole zpět ve směru útoku.
- **Wizard**
  - Životy: 2
  - Pohyb: pohyb do L jako šachový kůň
  - Útok: 1 pole dopředu
  - Schopnosti: Pokud se přemístí na figuru, prohodí si s ní místo.

- **Stomper**
  - Životy: 2
  - Pohyb: pohyb do L jako šachový kůň
  - Útok: Pokud se přemístí na figuru, rovnou ji zabije.
- **Cannon**
  - Životy: 2
  - Pohyb: 1 pole diagonálně
  - Útok: 3 pole diagonálně
  - Schopnosti: Útok se nezastaví o první figuru, ale prostřelí všechna 3 pole.
- **Ranger**
  - Životy: 1
  - Pohyb: 1 pole diagonálně
  - Útok: neomezeně polí diagonálně
- **Pawn**
  - Životy: 1
  - Pohyb: při jeho prvním tahu o 2 pole dopředu, potom o 1 pole.
  - Útok: 1 diagonální pole před sebou
  - Schopnosti: Pokud zabije figuru, přesune se na její místo. Pokud dojde na konec šachovnice, zabije libovolnou figuru včetně mastera.

4.2 Rozbor zdrojového kódu

V této části budou podrobně rozebrány a popsány jednotlivé třídy a rozhraní aplikace tak, aby čtenář mohl pochopit, jak se přesně při vývoji postupovalo. Kvůli přehlednosti a stručnosti budou vybrány jen podstatné části zdrojového kódu.

4.2.1 Class Bunka

Tato třída je velice důležitá pro tvorbu šachovnice. Do jejích instancí, do buněk, se ukládají důležité informace o jednotlivých polích.

Zdrojový kód 1: Konstruktor třídy Bunka

```
public Bunka(final int col, final int row) {
    this.sloupec = col;
    this.radek = row;
    this.figurka = null;
    this.barevnyCtverec = new Paint();
    this.selected = false;
    originalniBarva = (isDark() ? Color.BLACK : Color.WHITE);
    barevnyCtverec.setColor(originalniBarva);
}
```

Z konstruktoru této třídy lze vyčíst, jaké vlastnosti jednotlivé instance obsahují. Každá buňka má takto přiřazeno číslo řádku a sloupce, na kterém se v šachovnici nachází a to pak slouží jako souřadnice při definování její polohy. Dále jsou buňky naplněny figurkami, umí se vykreslovat a reagovat na dotek s tím, že si pamatují, jestli už byla vybrána nebo se na ni kliklo poprvé. Podle fáze výběru pak buňky mění svoji barvu, aby bylo poznat, jestli ji uživatel právě zvolil nebo ji odkliknul při výběru jiné. Toto řeší metoda **handleTouch()**.

Zdrojový kód 2: Obarvování buněk po kliknutí

```
public void handleTouch() {
    // nastaví barvy podle toho, jak byla buňka vybraná
    if(!selected){
        barevnyCtverec.setColor(Color.parseColor("#80123250"));
    }else{
        barevnyCtverec.setColor(originalniBarva);
    }
    // změní vybranost buňky podle předešlé vybranosti
     selected = !selected;
}
public boolean isDark() {
    return (sloupec + radek) % 2 == 0;
}
```

Metoda **isDark()** se pak stará o původní barvu buněk. Pokud je součet indexu sloupce a řádku dělitelný dvěma, vrátí pravdivou hodnotu oznamující, že buňka má černou barvu. Toto si lze snadno ověřit, jelikož první pole má na šachovnici být černé a index jeho řádku i sloupce se rovná jedné (1 + 1 = 2), tak se potvrdí podmínka dělitelnosti. S druhým políčkem však přichází zbytek po dělení a pole je tedy bílé (1 + 2 = 3).

Dále už tato třída obsahuje pouze důležité gettery a settery, které nastavují a vrací hodnoty, které se použijí v jiných částech projektu. To se týká hlavně proměnných, jako je figurka, souřadnice buňky a jejích okrajů, její barvy a vybranosti.

4.2.2 Class Sachovnice

Tato třída je spolu s Bunkou a Figurkou jedním z nejdůležitějších prvků tohoto projektu. Zde se totiž tvoří samotná herní plocha tvořená z dvojrozměrného pole buněk. Tyto buňky pak naplní nově vytvořenými figurkami a stará se i o vykreslení celé šachovnice a věcí, které se na ní udávají.

Zdrojový kód 3: Konstruktor třídy Sachovnice

```
public Sachovnice(final Context context, final AttributeSet attrs) {
    super(context, attrs);
    this.bunky = new Bunka[SLOUPCE][RADKY];
    instance = this;
    setFocusable(true);

    buildTiles();
    buildFigures();
    PraveVybranaFigurka.setOurInstance(null);
}
```

Z konstruktoru lze vyčíst, že šachovnice se skládá z dvojrozměrného pole buněk, jehož velikost je udána konstantami pro sloupce a řádky. Velikost těchto konstant je předem nastavena na osm (pole 8x8). Proměnná instance je zde, aby později zabránila vytváření dalších šachovnic. Metoda **buildTiles()** po zavolání konstruktoru na šachovnici vytvoří již zmíněné buňky a metoda **buildFigures()** je naplní novými figurkami.

Zdrojový kód 4: Vytvoření bílých věží

```
Context pomGame = this.getContext();
// casto volane požadavky:
final String frostTower = „Frost tower“
final String fireTower = „Fire tower“
final String helpFrostTower = pomGame.getString(R.string.helpFrostTower);
final String helpFireTower = pomGame.getString(R.string.helpFireTower);

bunky[0][0].setFigurka(new Figurka(frostTower, helpFrostTower, 3, true,
null, 0, 0));
bunky[7][0].setFigurka(new Figurka(fireTower, helpFireTower, 3, true, null,
7, 0));
```

Pro upřesnění, při tvoření všech osmi pěšáků se nevypisuje každý zvlášť, ale je zde použit for cyklus, který je postupně dosazuje na jejich pozice.

Zdrojový kód 5: Vytvoření černých pěšáků

```
for(int i = 0; i < 8; i++){
    bunky[i][6].setFigurka(new Figurka(pawn, helpPesak, 1, false,
null, i, 6));
}
```

Po vytvoření buněk a figurek se pomocí rozsáhlé metody **onDraw()** zjistí velikost displeje, ze které se pak vypočte optimální velikost čtverce na šachovnici a umístění jejího středu. Dále tato metoda vykreslí všechny dostupné možnosti pohybu figurky nebo je naopak nechá zmizet. Navíc se ještě stará o zobrazení počtu životů jednotlivých figurek a o vykreslení jejich obrázků. Tato metoda je velice důležitá, proto zde bude vypsána její velká část.

Zdrojový kód 6: Vykreslovací metoda onDraw()

```
protected void onDraw(final Canvas canvas) {
final int width = getWidth();
final int height = getHeight();

// výpočet velikosti čtverce na šachovnici
this.velikostCtverce =
Math.min(getSquareSizeWidth(width),getSquareSizeHeight(height));
// výpočet středu šachovnice
computeOrigins(width, height);
// vykreslení možných tahů figurky
if (!moznostiPohybu.isEmpty()){
    for (Bunka b: moznostiPohybu){
        b.getBarevnyCtverec().setColor(Color.GREEN);
    }
}else{
// překreslí všechny buňky do původní barvy
    for (int qw = 0; qw < SLOUPCE; qw++) {
        for (int wq = 0; wq < RADKY; wq++) {
bunky[qw][wq].getBarevnyCtverec().setColor(bunky().setColor(bunky
[qw][wq].getOriginalniBarva());
        }
    }
}
// vykreslí počet zdraví figurky
    textPaint.setColor(Color.GREEN);
    int pocetZ = bunky[c][r].getFigurka().getPocetZivotu();
    for (int i = 0; i < pocetZ; i++){
        canvas.drawRect(xCoord + 10, yCoord + 10 + (i*10), xCoord +
15, yCoord + 15 + (i*10), textPaint);
}
```

Z výše uvedeného kódu je zřejmé, že metoda **onDraw()** hlídá, jestli už právě nevykresluje možnosti pohybu nějaké figurky. Pokud ano, přebarví všechna vyznačená pole zpět na původní barvu, v opačném případě tyto možnosti zobrazí. Dohled nad počtem životů je vyřešen vykreslením zelených bodů v levé horní části buňky, kde se figurka nachází. Právě k tomu je důležité znát již dříve zmíněné souřadnice okrajů jednotlivých čtverců.

Další důležitou částí této třídy je metoda **provedPohyb()**, která řeší pohybovou stránku figurek. Zde je hlavním cílem roztřídit stav vybranosti buňky a řídit se podle toho zda se na ni už kliklo či nikoliv.

Zdrojový kód 7: Řešení pohybu figurek metodou provedPohyb()

```
private void provedPohyb(Bunka bunka) {
    if (PraveVybranaFigurka.getInstance() == null) {
        if (bunka.getFigurka() != null) {
            provedCisteni(bunka.getFigurka(), bunka);
            if (hrajeSpravnyHracFigur-
ky(PraveVybranaFigurka.getInstance()))
                bunka.getFigurka().zobrazPohyb(bunka);
        }
    }else{
        if (bunka.getFigurka()!=null){
            if (zkontrolujZdaKlikameNaTuJednuASamouBunkuSFigur-
kou(bunka)){
                provedCisteni(null, bunka);
            }
            provedCisteni(null, bunka);         }
        else{
            if (hrajeSpravnyHracFigur-
ky(PraveVybranaFigurka.getInstance())) {
                if (klikameNaMozneTahy(bunka)) {
                    vykonejPohyb(bunka);
}}}}}
```

Tato metoda je velice komplikovaná a při vývoji v ní lze udělat mnoho chyb. Je nutné ošetřit, jestli už je daná buňka vybraná, jestli je v ní nějaká figurka a kliká na ni její majitel nebo se omylem kliklo na prázdné pole. Zjednodušeně vysvětlený postup zní takto. Metoda nejprve zkoumá vybranost kliknuté buňky, pokud předem žádná vybrána není, zeptá se, jestli v ní je nějaká figurka. Pokud ano, vyčistí prostor, inicializuje figurku a kontroluje, zda patří hráči, který je na tahu. Při pravdivém výsledku se nakonec zeleně zobrazí možnosti pohybu. V případě, že již nějaká buňka vybrána byla a chceme s ní táhnout, zeptá se metoda, zda je kliknuto na jinou figurku nebo na prázdné pole. Při první možnosti se výběr resetuje, stav vybranosti se vyčistí a nastaví se na prázdnou hodnotu. V druhém případě se zkontroluje, jestli má figurka opravdu možnost stoupnout na vybrané prázdné pole. Pokud ano, vykoná se pohyb a přesune se tam. Výše vypsaný kód se ještě odkazuje na metodu **vykonejPohyb()**. Ta už je podstatně kratší a jednodušší. Má za úkol zjistit původní souřadnice figurky a

místa kam se má přesunout. Potom už jen vytvoří figurku v cíli, smaže její kopii na startu a předá tah druhému hráči.

Poslední důležitý úsek třídy **Sachovnice** je způsob určování souřadnic polí a metoda **provedCisteni()**, která má za úkol nastavit právě vybranou figurku, smazat seznam možností pohybu a vyžádat si překreslení šachovnice a levého panelu s nápovědou.

Zdrojový kód 8: metoda provedCisteni()

```
private void provedCisteni(Figurka f, Bunka bunka){
    PraveVybranaFigurka.setOurInstance(f);
    this.moznostiPohybu.clear();
    bunka.handleTouch(); // upraví vzhled buňky podle potřeby
    this.invalidate();
    LevyPanel.getInstance().invalidate();
}
```

Určování souřadnic je řešeno jednoduchým výpočtem, kdy se od celkové šířky displeje odečte šířka šachovnice. Tím po stranách zbyde prostor pro levý a pravý panel, a pokud je celý výsledek vydělen dvěma, vyjde souřadnice pro X v místě, kde začíná šachovnice. Výška je vyřešena stejným způsobem, a jelikož šachovnice sahá od spodní hrany obrazovky až po horní, je tedy původní Y rovno nule. Pak už se jen přičítají hrany čtverce, aby se docílilo identifikace jednotlivých buněk.

Zdrojový kód 9: Výpočet souřadnic jednotlivých buněk

```
private void computeOrigins(final int width, final int height) {
    this.x0 = (width  - velikostCtverce * 8) / 2;
    this.y0 = (height - velikostCtverce * 8) / 2;
}
private int getXCoord(final int x) {
    return x0 + velikostCtverce * (flipped ? 7 - x : x);
}
private int getYCoord(final int y) {
    return y0 + velikostCtverce * (flipped ? y : 7 - y);
}
```

Takto vypadá zobrazená šachovnice. Její parametry se sice nastavují v této třídě, ale samotné vykreslení probíhá ve třídě Game, která bude popsána později.

Obrázek 8: Vykreslená šachovnice

4.2.3 Class Figurka

Třída **Figurka** je asi nejrozsáhlejší částí tohoto projektu. Je zde definován konstruktor, pomocí kterého jsou tvořeny všechny figurky a dále dvě důležité metody, jedna pro řízení pohybu a druhá pro řízení útoku. Tyto metody jsou velice náročné a rozsáhlé, jen řešení pohybu všech figurek zabere přibližně šest set řádků kódu.

Zdrojový kód 10: Konstruktor třídy Figurka

```
Figurka(String t, String np, int pZ, Drawable v,  boolean str, int x,
int y){
    this.typ = t;
    this.napoveda = np;
    this.pocetZivotu = pZ;
    this.vzhled = v;
    this.strana = str;
    this.x = x;
    this.y = y;
}
```

Tvorba figurek už byla nastíněna na předešlých ukázkách kódu. Zde se čtenář může dozvědět, jaké parametry přesně každá figurka přebírá. Dědí se její typ (master, pawn), nápověda, do které se ukládá text poskytující informace o figurce, počet životů, vzhled v podobě obrázku, barva strany hráče (černá, bílá) a souřadnice buňky, ve které se figurka nachází.

Možnost pohybu je vyřešena souborem podmínek, které vymezují pole, na něž může daná figurka podle pravidel stoupnout. Tato pole se určují pomocí souřadnic řádků a sloupců buněk, kde se jako první bere buňka v levém dolním rohu, která má index [0][0]. Je důležité si uvědomit, že definice pohybu všech figurek není pro obě strany stejná. Touto komplikací jsou pěšáci, kteří mohou jít pouze směrem dopředu. Dalším problémem k vyřešení je ošetřit, aby žádná z figurek nevylézala ze šachovnice. Toho je docíleno podmínkami, které určují, jakých hodnot indexů může pozice nabýt.

Pawn

Zdrojový kód 11: Řešení pohybu černého pěšáka

```
public boolean zobrazPohyb(Bunka bunka) {
    if(strana){
        switch (typ){
            case "Pawn":
                if (bunka.getRadek() < 7){
                    if (Sachovni-
ce.getInstance().getBunky()[bunka.getSloupec()][bunka.getRadek()+1].g
etFigurka()==null)
            Sachovni-
ce.getInstance().getMoznostiPohybu().add(Sachovnice.getInstance().get
Bunky()[bunka.getSloupec()][bunka.getRadek()+1]);
                if (!hnulSe)
                    if (Sachovni-
ce.getInstance().getBunky()[bunka.getSloupec()][bunka.getRadek()+2].g
etFigurka()==null)
            Sachovni-
ce.getInstance().getMoznostiPohybu().add(Sachovnice.getInstance().get
Bunky()[bunka.getSloupec()][bunka.getRadek()+2]);
                }
            break;
```

Zde je možno vidět jak metoda ***zobrazPohyb()*** pracuje. Nejdříve se zeptá, k jaké straně daná figurka patří a dále zkoumá, o jaký typ jde, aby mohla vyřešit možnosti jejího pohybu. V této části kódu se jedná o pohyb černého pěšáka, u kterého je nutno ošetřit aby se mohl první tah posunout o dvě pole, ale další kola už jen o jedno. Zde je ale vyřešen pouze pohyb černé strany, protože pěšák může pouze dopředu, navyšuje se souřadnice řádku o jedna nebo o dva. Pokud by se jednalo o bílou stranu, jejíž pěšáci začínají na řádku s indexem 2 a postupují směrem nahoru, tak by se souřadnice řádku snižovala.

(Pro stručnost už budou dále uváděny pouze změny souřadnic figurek a bude vynechán opakující se kód:

*S*achovnice.*getInstance*().getMoznostiPohybu().add(Sachovnice.*getInstance().getBunky())*

Zdrojový kód 12: Ukázka útoku bílého pěšáka

```
public boolean zobrazUtok(Bunka bunka) {
switch (typ){
case "Pěšák":
if(strana) { // pokud patří bílému hráči
 if (bunka.getRadek() < 7){ //pokud ještě nedošel na okraj šachovnice
  if([bunka.getSloupec()-1][bunka.getRadek()+1].getFigurka()!= null
&&[bunka.getSloupec()-1][bunka.getRadek()+1].getFigurka().isStrana()
!= strana){
Sachovnice.getInstance().setVybratSiCilUtoku(true);// povolí útočení
Sachovnice.getInstance().getMoznostiPohybu().add(// přidá danou pozi-
ci do útoku
Sachovnice.getInstance().getBunky()[bunka.getSloupec() -
1][bunka.getRadek() + 1]); }
```

V této ukázce je část metody *zobrazUtok()*, která zde řeší možnosti útoku bílého pěšáka v diagonálním směru doprava nahoru. Jelikož hráč nejspíš nebude chtít ničit vlastní figurky, je zde přidána podmínka, která zjišťuje barvu vybraného pěšáka a figurky, na níž má zaútočit. Další podmínka pustí proces dále, pouze pokud je na prvním poli v tomto směru nějaká soupeřova postava. V případě, že jsou všechny podmínky splněny, přidá metoda pole do seznamu možností a povolí na něj zaútočit.

Frost tower

U pohybu ledové věže se vyskytl problém s překračováním jiných figurek. Jelikož se může hýbat až o dvě pole, může nastat situace, kdy přímo před věží stojí nějaká figurka, ale další pole už je volné. Je tedy nutné ošetřit, aby se Frost tower zastavila před překážkou a nemohla za ní pokračovat. To je vyřešeno dotazem, jestli v daném směru nestojí nic v cestě, a pokud ano, má zakázáno pokračovat.

Zdrojový kód 13: Řešení pohybu Frost tower, směrem dolu

```
case "FrostTower":
    boolean muzuDoluVez = false;
    boolean muzuNahoruVez = false;
    boolean muzuDopravaVez = false;
    boolean muzuDolevaVez = false;
// směr o 1 dolu
if (bunka.getSloupec() < 7){ // ošetření, aby nevylezla ze šachovnice
   [bunka.getSloupec() + 1][bunka.getRadek()]);
 if([bunka.getSloupec() + 1][bunka.getRadek()].getFigurka() == null){
        muzuDoluVez = true;
    }
}// o 2 dolu
if(muzuDoluVez){
 if (bunka.getSloupec() < 6) {
    [bunka.getSloupec() + 2][bunka.getRadek()]);
    muzuDoluVez = false;
}}
```

Fire tower

Ohnivá věž je na naprogramování mnohem snazší než její ledová sestra. Jelikož se hýbe pouze o jedno pole, lze převzít první polovinu pohybu z Frost turret a je hotovo. Možnosti útoku také nijak komplikované nejsou, její dosah jsou tři rovná pole a zasáhne pouze první figurku v daném směru.

Stomper

U řešení pohybu Stompera, který se hýbe stejně jako kůň v klasické verzi šachů, se lze velice snadno ztratit a udělat chybu. Zvláštní pohyb ve tvaru písmene L nabízí osm možných pozic, kam se může přesunout. Problémem je ošetřit všechny okraje šachovnice, aby z ní kůň nevylezl a nezpůsobil tím chybu.

Zdrojový kód 14: Ukázka řešení pohybu Stompera

```
case "Stomper":
if (bunka.getSloupec() < 6 && bunka.getRadek() < 7) { // dolu doprava
[bunka.getSloupec() + 2][bunka.getRadek() + 1]);
}
if (bunka.getSloupec() < 7 && bunka.getRadek() < 6) { // doprava dolu
[bunka.getSloupec() + 1][bunka.getRadek() + 2]);
}
if (bunka.getSloupec() > 0 && bunka.getRadek() < 6){ //doprava nahoru
[bunka.getSloupec() - 1][bunka.getRadek() + 2]);
}
// a dalších 5 směrů...
break;
```

Wizard

Pohyb Wizarda je úplně stejný jako pohyb Stompera, liší se však svým útokem a hlavně schopností. Wizard útočí na jedno pole směrem k soupeři, na tom není nic složitého k naprogramování. Jeho schopnost vyměnit si pozici s jakoukoliv figurkou, na kterou stoupne, je už náročnější. Problém je v tom, že se musí udělat výjimka v herním systému, kde při kliknutí na jinou figurku zmizí možnosti pohybu a označí se druhá figurka. Wizarda lze tedy odznačit pouze kliknutím na prázdné pole.

Blade dancer

Tato figurka má asi nejsilnější schopnosti ze hry. Může se totiž pohybovat neomezeně polí diagonálně i axiálně a svým útokem zasáhne všechno kolem sebe. Neomezený pohyb však s sebou přináší problém s vylézáním ze šachovnice, který je nutno vyřešit. Toto je ošetřeno for cyklem a podmínkami, kte-

ré upravují nejen hranici dostupnosti, ale řeší i zákaz přeskakování ostatních figurek stojících v cestě.

Zdrojový kód 15: Ukázka řešení pohybu figurky Blade dancer

```
case "BladeDancer":
boolean muzuDoluDoleva = true;
for (int x = 1; x < 8; x++) {
// diagonální směr dolu doleva
   if (bunka.getSloupec() < (8 - x) && bunka.getRadek() > (-1 + x)) {
        if (muzuDoluDoleva) {
            if (Sachovni-
ce.getInstance().getBunky()[bunka.getSloupec() + x][bunka.getRadek()
- x].getFigurka() != null) {
                muzuDoluDoleva = false;
            }
      ..............[bunka.getSloupec() + x][bunka.getRadek() - x]);
        }
    }
muzuDoluDoleva = true;
break;
```

Ve výše uvedeném kódu je uvedena ukázka řešení pohybu pouze jednoho směru. Nejdříve je stanovena booleanovská hodnota určující, jestli může Blade dancer pokračovat v cestě. Ta zůstane pravdivá, pouze pokud na zkoumané buňce nestojí žádná jiná figurka. Jinými slovy se nastaví na false v případě, když je hodnota figurky na poli v cestě nenulová. Pokud se těmito podmínkami projde, přidá se zkoumané pole do seznamu možností pohybu. Nakonec se pravdivostní hodnota nastaví na true, pro příští tahy.

4.2.4 Class LevyPanel

Tato třída se stará o zobrazování informací a nápovědy v levém panelu herní obrazovky. Hráč zde po kliknutí na figurku může vidět obrázek s vyznačenými políčky, na které se může pohnout nebo zaútočit. Dále zde bude slovní popis figurky, jejích schopností nebo aktuální stav jejího zdraví a také informace sdělující, který hráč je na tahu. To vše zde zpracovává jediná rozsáhlá metoda **onDraw()**, která připravuje vykreslovací komponenty a zjišťuje všechny informace potřebné k zobrazení zmíněných údajů.

Zdrojový kód 16: Výpis názvu figurky s metodou onDraw()

```
protected void onDraw(Canvas canvas) {
super.onDraw(canvas); // připraví si nástroje na kreslení
Paint textPaint = new Paint();
textPaint.setARGB(150, 000, 000, 000); // nastavení barvy písma
textPaint.setTextAlign(Paint.Align.CENTER); // středové zarovnání
textPaint.setTextSize(13);
canvas.drawText("Selected figure: ", (canvas.getWidth() / 2), 25,
textPaint);
textPaint.setTextSize(30);
    // vypíše právě vybraný typ figurky
    if (PraveVybranaFigurka.getInstance()!=null){
        canvas.drawText(PraveVybranaFigurka.getInstance().getTyp(),
(canvas.getWidth() / 2), 60, textPaint);
}
```

Metoda onDraw() si nejprve připraví nástroje na vykreslování. Poté nastaví všechno potřebné pro úpravu textu. Tyto parametry se týkají nastavení jeho barvy, velikosti, zarovnání a jeho náplně, která se bude zobrazovat. Výše vybraná ukázka nejprve vypíše frázi „Selected figure:“, pod kterou se písmem o velikosti třicet pixelů přidá text s názvem právě kliknuté figurky. Zbytek informací pro levý panel se zobrazuje podobně, je tedy zbytečné tuto metodu dále rozvádět.

4.2.5 Class Game

V této třídě se inicializují tlačítka pro pravý panel herní obrazovky a také se zde volá velice důležitá aktivita **aktivity_game**, ve které jsou obsaženy všechny důležité parametry pro založení obou panelů a šachovnice. Tato aktivita se nachází v adresáři pro layout, do kterého se ukládají aktivity pro zobrazování uživatelského prostředí. Jako tlačítka byly vybrány čtyři funkce, otočení obrazovky o 180 stupňů, vrácení do hlavního menu, vrácení tahu a posunutí tahu zpět dopředu. Pro možnost demonstrace zobrazování rychlých připomínek od telefonu se u některých těchto tlačítek ukáže zpráva, že funkci je nutno dokoupit v obchodě.

Zdrojový kód 17: Volání hlavní herní aktivity a zobrazení rychlé nápovědy

```
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
setContentView(R.layout.activity_game);
game = this;
initBt();
}
private void initBt() {
ImageButton back = (ImageButton) findViewById(R.id.btBackTurn);
back.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
Toast.makeText(game, "You can buy this function in our shop.",
Toast.LENGTH_SHORT).show();
}}); }
```

V této ukázce je zobrazena metoda **onCreate()**, která otáčí obrazovku do vodorovné polohy, volá již zmíněnou layout aktivitu a také metodu **initBt()**. Ta slouží pro definování funkce tlačítek po jejich stisknutí. V kódu se nachází důležité slovo Toast, které slouží pro zavolání rychlé nápovědy. V tomto případě se po kliknutí na tlačítko pro vrácení tahu na malou chvilku zobrazí textová zpráva ve tvaru „You can buy this function in our shop.“.

Zdrojový kód 18: Ukázka inicializace tlačítka v acticity_game

```
<ImageButton
    android:id="@+id/btBackTurn"
    android:layout_weight="1"
    android:src="@drawable/ic_btback"
    android:padding="20dip"
    android:gravity="center"
    android:enabled="true" />
```

V této části probíhá inicializace tlačítka pro vrácení tahu a nastavení jeho parametrů. Po zavolání aktivity se tento kód spustí a provede svou funkci. Po prozkoumání ukázky si lze všimnout odkazu na adresář drawable, který byl zmíněn v teoretické části v souvislosti s ukládáním obrázků.

4.2.6 Class PravyPanel

Jelikož je o vykreslování všech částí pravého panelu už postaráno, slouží tato třída jako pouze jako pomocník při zakládání konstruktoru, který volá potřebné atributy.

O tlačítkách a jejich tvorbě již bylo všechno podstatné zmíněno výše, ale stále je zde jedna část k rozboru a tou je prostor pro reklamy. Tato práce neslouží ke komerčním účelům, a proto nejsou skutečné reklamy v aplikaci zahrnuty. Pro demonstrační účely však bude vysvětlen obecný postup, jak je do své aplikace přidat.

Zakomponování reklam do aplikace

Reklamy se podle zobrazení dělí na dva typy. Pokud jde o menší plochu zobrazenou někde u kraje displeje, jedná se o tzv. banner. Tento typ se používá zejména u aplikací, které mohou nabídnout část svého prostoru, aniž by tím nějak omezovaly jejich funkcionalitu. Další typ se zobrazuje na celou obrazovku, většinou v případě nějaké akce vyvolané uživatelem, jako je pauza nebo spuštění nové hry. Toto se nazývá vsunutá reklama. Obecný postup jejich zahrnutí by v kódu vypadal asi takto.

Zdrojový kód 19: Obecný postup při vkládání reklamy

```
1.) private InterstitialAd mInterstitialAd;
    mInterstitialAd = new InterstitialAd(this);
    mInterstitialAd.setAdUnitId("ca-app-pub-39402562544/1033173712");
2.) AdRequest adRequest = new AdRequest.Builder()
    .addTestDevice("YOUR_DEVICE_HASH")
    .build();
     mInterstitialAd.loadAd(adRequest);

3.) if (mInterstitialAd.isLoaded()) {
    mInterstitialAd.show();}
4.) xmlns:ads="http://schemas.android.com/apk/res-auto"

5.) AdView mAdView = (AdView) findViewById(R.id.adView);
     AdRequest adRequest = new AdRequest.Builder()
    .addTestDevice("YOUR_DEVICE_HASH")
    .build();
    mAdView.loadAd(adRequest);
```

Prvním krokem je vytvořit objekt pro vsunutou reklamu a přiřadit mu jeho ID. Dále je nutné vyžádat si načtení reklamy, a pokud proběhlo v pořádku, bude reklama vykreslena. V případě, že jde o tvorbu banneru, jsou jeho atributy definovány v XML layout adresáři, pak už ho stačí jen načíst stejným způsobem jako v první části kódu.

4.2.7 Class Menu

Tato třída a její stejnojmenná aktivita mají na starosti zobrazování a funkcionalitu hlavního menu, které uživatel vidí po zapnutí aplikace. Toto menu se skládá z pozadí tvořeného kruhovým obrázkem s motivem dřeva. Na kruhu se nachází tlačítka v podobě kamenných kusů, které se po kliknutí lehce rozdělí, aby došlo k estetickému efektu tříštění. To celé pak přechází do černého pozadí, které vždy vyplní zbytek obrazovky, tím se docílí lehké nezávislosti na velikosti displeje zařízení. Funkce tlačítek jsou založení nové hry, vrácení se do hry, nastavení, zobrazení informací o hře, ukončení aplikace a obchod, který je v této verzi uzavřen, aby nedocházelo ke komercializaci.

Zdrojový kód 20: Tvorba tlačítka pro ukončení aplikace

```
private Button konec;
private void setFceButtons(){
    konec = (Button) findViewById(R.id.button_konec);
    konec.setOnClickListener(new View.OnClickListener() {

        @Override
        public void onClick(View v) {
            if(v == konec) {
                konec.setBackgroundDrawable (getRe
sources().getDrawable(R.drawable.button_onclick));
            }
            finish();
            System.exit(0);
    } });}
```

Po založení proměnné pro tlačítko je do ní přes jeho ID přiřazena hodnota s konkrétními atributy. Místo kde se tyto hodnoty přiřazují je layout aktivita **aktivity_menu**, která obsahuje inicializaci všech tlačítek z této nabídky. Pro upřesnění text s informacemi o hře, který se zobrazí po stisknutí tlačítka About

je uložen v adresáři strings.xml ve složce values. Tam jsou podle správných zvyklostí android vývoje uloženy všechny důležité textové řetězce např. nápověda k figurkám nebo jejich názvy. Slova uprostřed tlačítek však mohou být definována už ve zmíněné aktivitě, ale v tomto případě jsou jen součástí předem připraveného obrázku. Zajímavý je efekt tříštění tlačítek po kliknutí na ně, kterého je docíleno zobrazením jiného už rozbitého obrázku.

Obrázek 9: Ukázka změny tlačítka po jeho stisknutí

4.2.8 Class About

Tato krátká třída se stará pouze o nastavení obrazovky po kliknutí na tlačítko About v hlavním menu. V její jediné metodě **onCreate()** se volá aktivita **activity_about** a nastavuje se obrazovka do vodorovné polohy. Ve stejnojmenné aktivitě se hlavně upravuje text s informacemi o hře.

Zdrojový kód 21: Úprava textu v activity_about

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:textAppearance="?android:attr/textAppearanceLarge"
    android:layout_gravity="center_horizontal"
    android:text="@string/autor"
    android:layout_weight="0.14" />
```

V ukázce si lze povšimnout, že samotný textový řetězec, který se má zobrazit, není definován zde, ale je na něj opět odkázáno do již zmíněného adresáře Strings ve složce Values. Tyto řetězce se pak zobrazí jako přehled informací o hře a autorovi.

4.2.9 Class Setting

Jak už název napovídá, tato třída má za úkol zprovoznit některá nastavení hry. Po kliknutí na tlačítko Settings se zobrazí protokol, kde si uživatel může nastavit vlastní jméno obou hráčů, povolenou délku tahu nebo jazyk aplikace. Pro demonstrační účely je v této verzi předem nastavená angličtina a možnost výběru je zašedlá. To poslouží jako názorný příklad odepření přístupu uživateli, kdyby to bylo v budoucnu potřeba. Zmíněný nastavovací formulář je vytvořen přes nastavení preferencí v xml adresáři ve složce preferences.xml. Samotná třída už jen zavolá metodu **onCreate()**, která se postará o zařazení těchto preferencí do aplikace.

Zdrojový kód 22: Ukázka volání preferencí a jejich nastavení

```
public void onCreate(final Bundle savedInstanceState)
{
    super.onCreate(savedInstanceState);
    addPreferencesFromResource(R.xml.preferences);
}

// nastavení zvoleného jazyka ve složce preferences.xml
<ListPreference     android:title="Chosen language"
    android:summary="@string/language"
    android:enabled="false"
    android:key="language"
    android:defaultValue="1"
    android:entries="@array/listJazyk"
    android:entryValues="@array/jazykValue" />
```

V této ukázce je přiblíženo, jak se dostat do složky s preferencemi a zavolat z ní potřebné zdroje. Dále zde bylo vybráno zmíněné nastavení jazyka, ke kterému byl uživateli odepřen přístup pomocí přiřazení false hodnoty do enabled parametru. Toto ponechá zobrazení informace o tom, že je nastavená angličtina, ale znemožní jakýkoliv zásah pro další nastavení.

4.2.10 Class PraveVybranaFigurka

Tato pomocná třída dědící od jejího předka **Figurka** má na starosti pouze vytváření instance pro vybranou figurku. To je užitečné v jiných částech programu, kde je potřeba se odvolat na figurku, na kterou hráč právě kliknul. Potom co aplikace získá tuto informaci, může s ní např. zaútočit nebo vykreslit její pohyb. Kód této třídy je jednoduchý.

Zdrojový kód 23: Vytvoření instance právě vybrané figurky

```
public class PraveVybranaFigurka extends Figurka{
    private static Figurka ourInstance = new PraveVybranaFigurka();
    private PraveVybranaFigurka() {
        super();
        ourInstance = null;
    }
    public static void setOurInstance(Figurka f){
        ourInstance = f;
    }
    public static Figurka getInstance() {
        return ourInstance;
    }
}
```

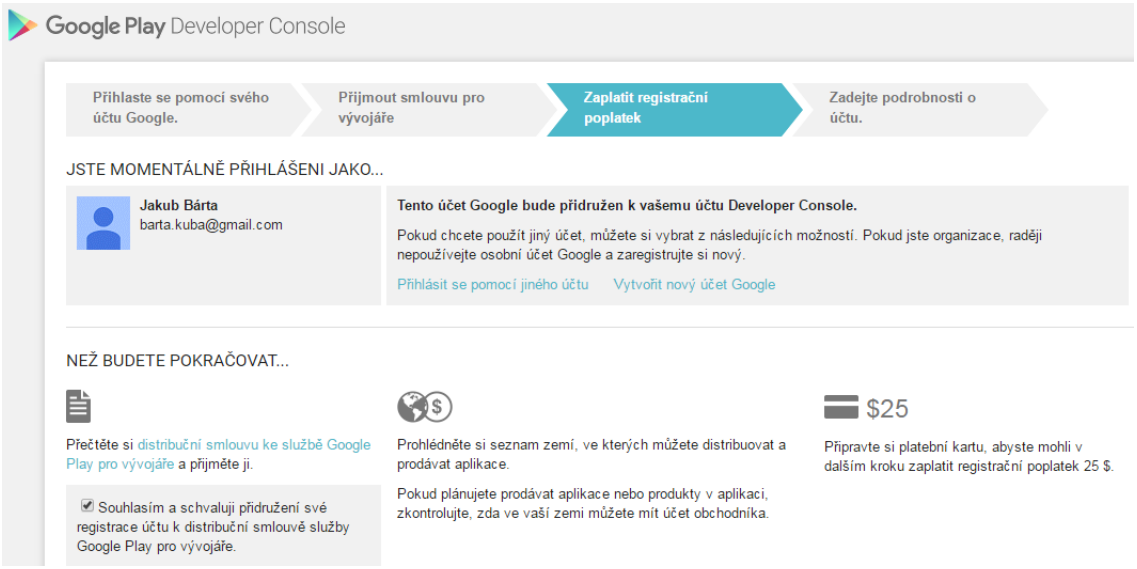
4.2.11 Interface IfaceFigurka

Toto jediné rozhraní obsahuje pouze dvě booleanovské metody ***zobrazPohyb()*** a **zobrazUtok()**, které jsou implementovány do třídy **Figurka**, kde řeší zobrazení možností pohybu a útoku.

4.3 Popis uvedení aplikace na trh

Po jejím dokončení bude aplikace umístěna na trh, ale jak vyplývá z postupů zjištěných při literárním průzkumu, je předtím potřeba učinit několik kroků. Prvním je zjistit si, jak celý proces funguje a jaké právní náležitosti s sebou přináší. To vše lze vyčíst z oficiálních internetových stránek pro Android vývojáře[3], kde je všechno podrobně popsané. Dalším krokem je vytvořit si vývojářský účet na stránkách *https://play.google.com/apps/publish/signup/* .

Obrázek 10: Registrace vývojářského účtu pro Google play

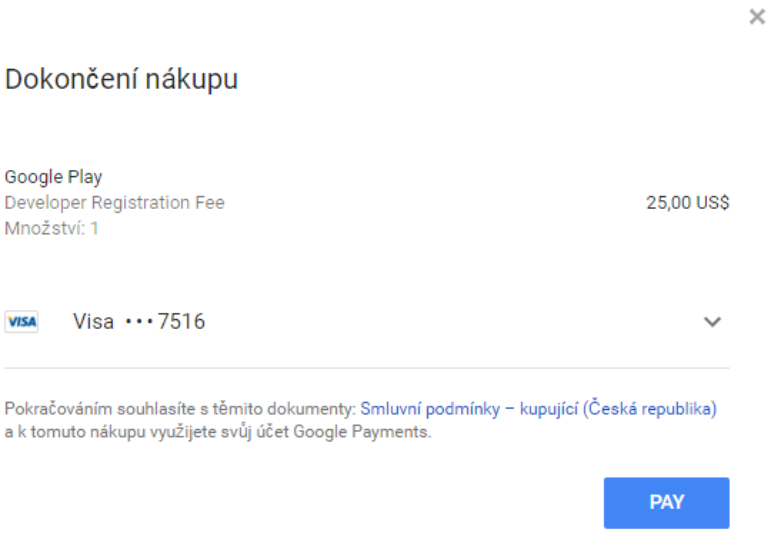


(Google, © 2016)

Po pročtení smlouvy o distribuci a odsouhlasení všech podmínek se zobrazí formulář pro uhrazení povinného poplatku ve výši 25 dolarů. Tento poplatek je jednorázový a hlavně permanentní. Je ho tedy nutné uhradit pouze jednou pro daný účet. Toto se může u jiných společností lišit, protože některé z nich požadují měsíční nebo roční předplatné.

[3] http://developer.android.com/distribute

Obrázek 11: Protokol pro úhradu povinného poplatku 25$

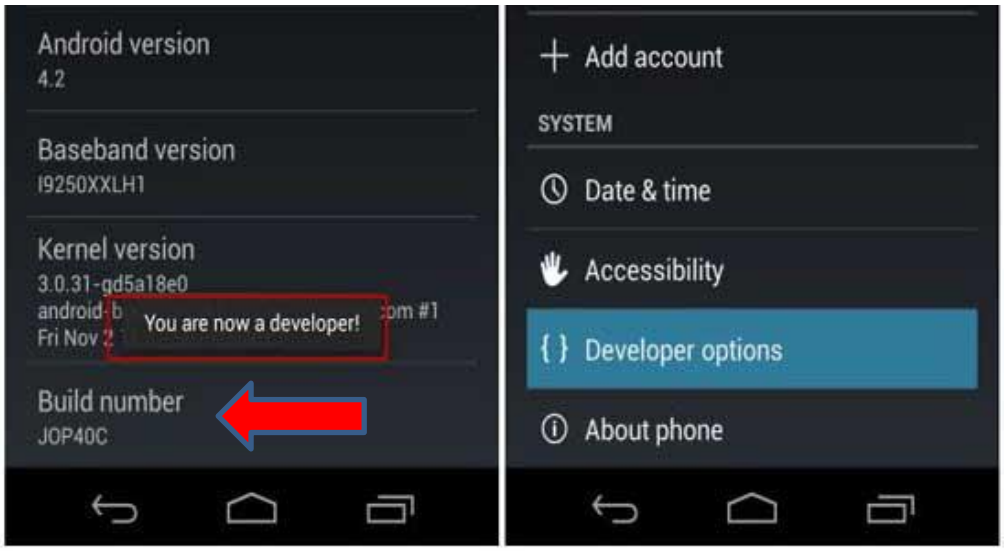


(Google, © 2016)

Po uhrazení poplatku, už stačí jen zadat podrobnosti o účtu a jeho tvorba je kompletní. Před publikováním aplikace je doporučeno ji nejdříve řádně otestovat. Ideální je nainstalovat ji na co největší počet zařízení a zkusit všechny možnosti, které uživatel má např. kontrola, jestli nevadí zapsání háčků a čárek či jiných znaků do jména hráče. Jestli se celá aplikace správně zobrazuje a funguje jak by měla, lze otestovat i připojením zařízení přes USB použít ho jako emulátor. Tato možnost je mnohem rychlejší než využití emulátorů v Android studiu, protože jejich simulace potřebují hodně výkonu a výrazně zpomalují chod počítače.

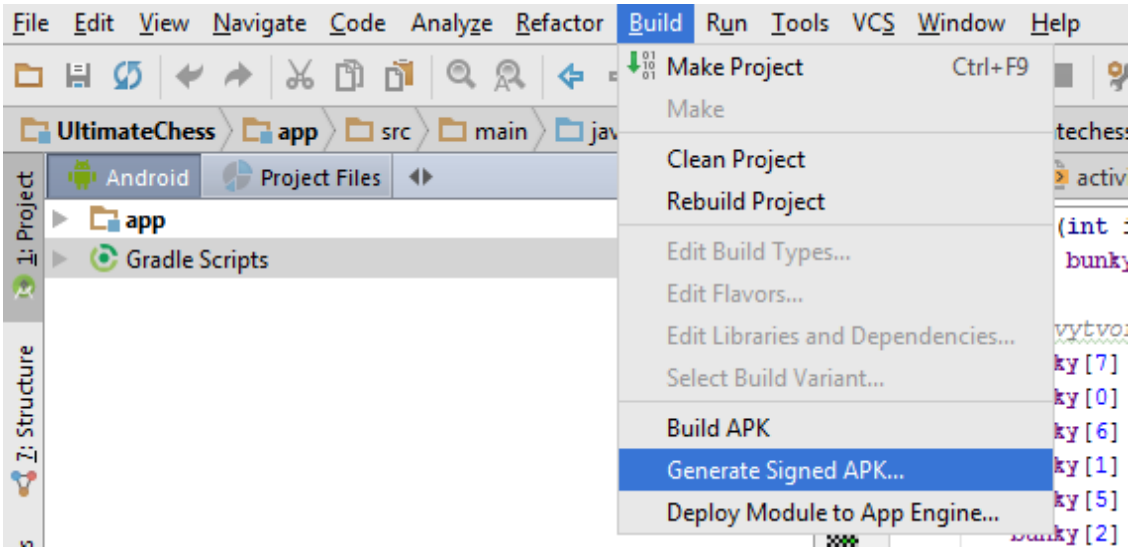
U simulací na chytrých telefonech se však musí provést několik nastavení. Nejprve musí majitel zařízení povolit možnosti vývojáře. K tomu vede cesta: *nastavení -> informace o telefonu -> softwarové informace -> číslo sestavení*. Pokud uživatel následně sedmkrát klikne na číslo sestavení, zobrazí se zpráva, že se stal vývojářem a možnosti pro vývoj jsou odemčeny. Pak už jen stačí povolit ladění přes USB a je hotovo.

Obrázek 12: Povolení vývojářských možností v telefonu



Dalším krokem po otestování aplikace je vytvoření jejího APK[4] souboru. Tento formát slouží podobně jako .exe u počítačů a lze přes něj tedy spouštět a nainstalovat finální program. Pro tvorbu APK souboru je nutno nejdříve zkontrolovat, jestli v kódu nejsou nějaké závažné chyby, protože by celá kompilace jinak neproběhla v pořádku. Po kliknutí na tlačítko Build v horním panelu Android studia, se zobrazí možnosti, z nichž se vybere Generate Signed APK…

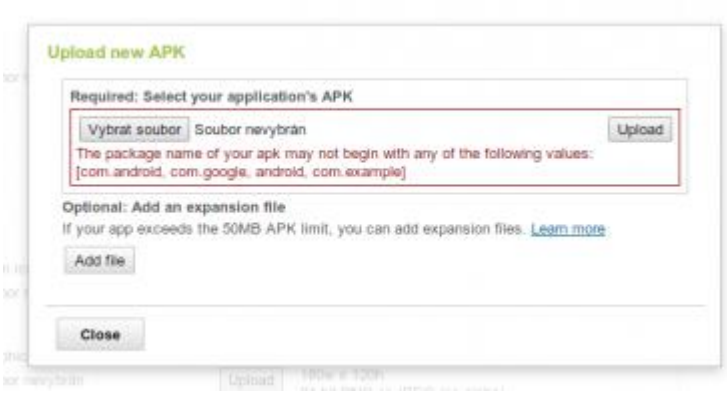
Obrázek 13: Generování APK souboru v Android studiu



[4] APK - Android application package

Následně se zobrazí několik formulářů, které je třeba vyplnit. První se týká vytvoření souboru s privátními klíči (Key store). Tento soubor slouží jako podepsání své aplikace autorem a je užitečné si ho dobře uložit. Vyplňuje se zde jeho jméno, heslo a cesta, kam se má uložit. Dále se uživatel musí přihlásit svým nastaveným heslem a zvolit cestu pro vytvoření APK souboru. Pokud vše proběhlo v pořádku, může se tento soubor umístit na Google play. Hned na úvodní stránce vývojářského účtu se nachází tlačítko Upload Application, které vede k nahrání vytvořeného APK.

Obrázek 14: Načtení APK souboru na Google play

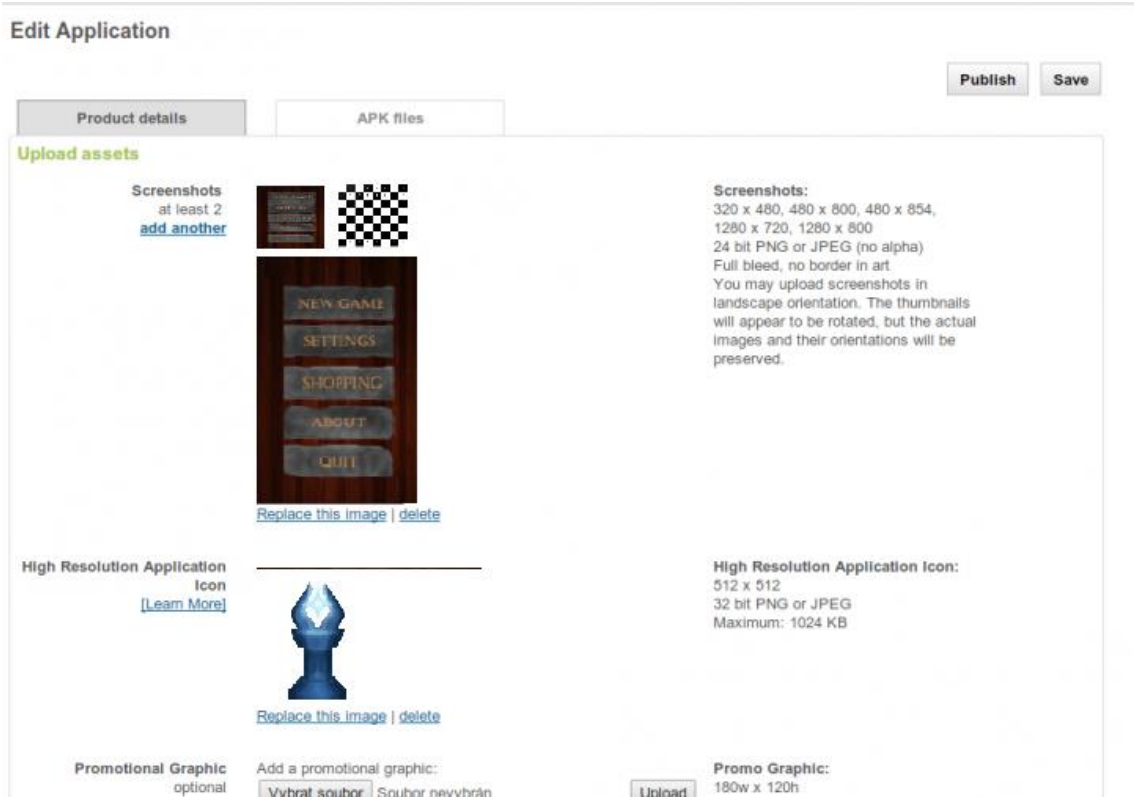


(Google, © 2016)

Po kliknutí na tlačítko Add file a načtení vybrané aplikace, se ještě nastaví jméno její verze a požadavky, které budou muset být povoleny budoucími uživateli, aby mohli program spustit. Aplikace se často ptají např. na povolení pro určení polohy nebo některé informace z telefonu.

Dále se nastavují detaily aplikace. Google play požaduje přidání alespoň dvou snímků přímo z aplikace, aby měl potenciální zákazník představu, jak vypadá její prostředí. Dále se zde nastavuje obrázek spouštěcí ikony, reklamní grafika a také kvalita všech snímků. Reklamní grafika slouží např. jako obrázek, který se ukáže v seznamu doporučených aplikací v Play store nebo jako odkaz na video určené k upoutání nových uživatelů. Další v pořadí je nastavení práv, které se hodí např. při propagaci samotným Googlem. Dále je zde výběr jazyků aplikace, vypsání titulku a jejího stručného popisu. Tam by neměly chybět informace o tom, k čemu aplikace slouží a také lákadla pro větší počet stažení. Jako další je možné vyplnit okno s popisem nedávných změn nebo přidat reklamní text.

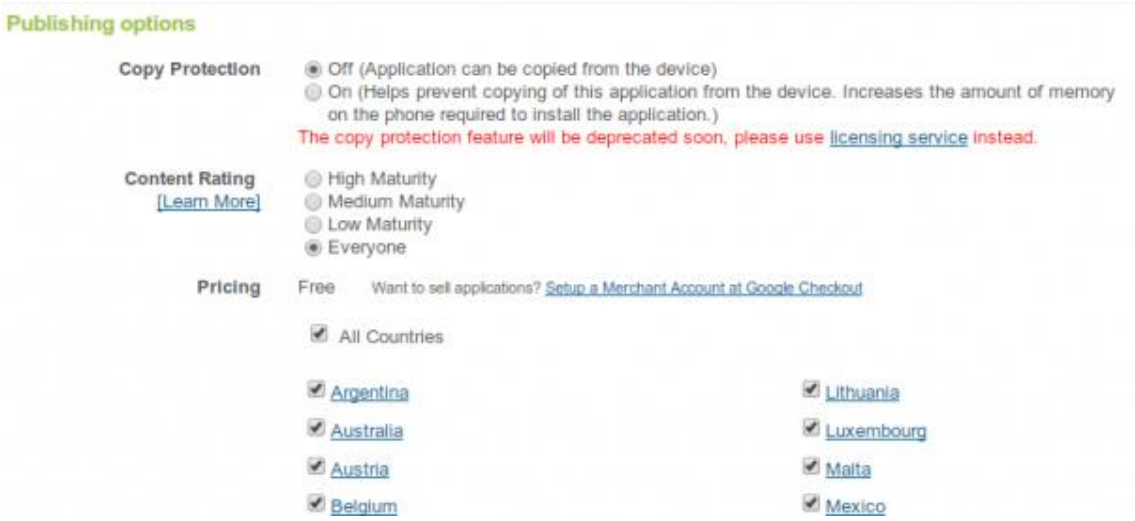
Obrázek 15: Nastavení detailů aplikace na Google play



(Google, © 2016)

Po nastavení informací o aplikaci se ještě musí vybrat kategorie, do které spadá, dále jestli je vyžadována ochrana proti kopírování, světové regiony, kde bude aplikace k dispozici ke stažení, a jestli je aplikace zdarma popř. její cena. Nakonec už se jen potvrdí podmínky Googlu pro distribuci aplikace a vše se pro jistotu uloží.

Obrázek 16: Nastavení distribuce aplikace na Google play



(Google, © 2016)

Při zaškrtnutí políčka, že aplikace bude k dispozici zdarma, se ještě zobrazí okno s dotazem, jestli s tím opravdu poskytovatel souhlasí, a že už to v budoucnu nepůjde změnit. Po úspěšném nastavení všech parametrů se ještě musí aplikace aktivovat. V záložce APK files se vybere právě nainstalovaná verze a klikne se na tlačítko Activate. Potom se vše musí znovu uložit, následně kliknout na Publish a ještě jednou odsouhlasit, že produkt je nabízen zdarma. Tím je aplikace hotová a zveřejněná na trhu.

Obrázek 17: Konečná fáze publikování aplikace na Google play

(Google, © 2016)

Na Google play se však nezobrazí hned, ale až po určité prodlevě, která může trvat i několik dní. Pokud chce její poskytovatel později aplikaci nějakým způsobem upravit, musí upravit její kód, poté ji znovu uložit pod stejným klíčem a heslem jako předchozí verzi, ale s rozdílem zvýšení čísla verze o jedna. Jinak se nevyhne upozorněním a chybovým hláškám od Googlu. Aplikace se znovu nahraje a uloží. Ve složce APK files pak budou k nalezení dva soubory. Po výběru toho novějšího a jeho aktivaci bude provedena požadovaná aktualizace. Nakonec je zvykem vypsat změny a novinky, které nahraná verze přináší. K tomu stačí najít tlačítko Product details, do pole Recent changes vypsat všechny podrobnosti a vše samozřejmě uložit.

Platby

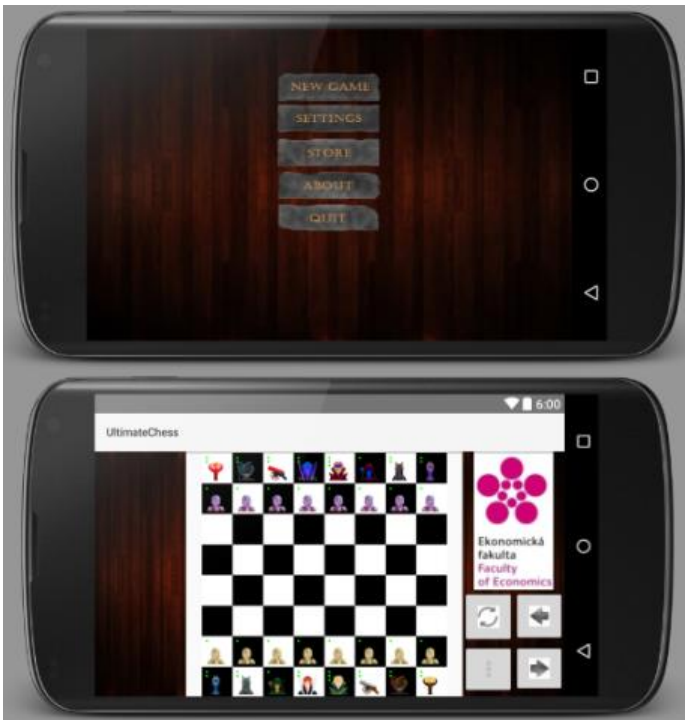
Pokud chce vydavatel za stažení své aplikace obdržet nějakou částku, musí si zřídit a udržovat tzv. platební účet u autorizovaného zpracovatele plateb. To lze zařídit na stránkách Google checkout. Velice praktické je, že lze propojit Google účet s bankovním. Jeho majitel se tedy nemusí starat o žádné složité převody tržeb. Při určování ceny za nabízený produkt je důležité počítat s transakčním poplatkem, který Google odečte od celkové částky. Tento poplatek, který je Google oprávněn změnit, činí v dnešní době 30% z ceny. Zbylých 70% dostane vydavatel aplikace. Kromě platby za stažení lze získat tržby prodejem produktů přímo v aplikaci. O vyúčtování za různá rozšíření nebo jiné zboží se postará služba Inn-app billing, která je navržena tak, aby samotná aplikace nemusela nijak řešit transakční procesy. Produkty lze prodávat i v cizích měnách. Google však neručí za správnost měnových kurzů a jejich převodu a má právo si přiúčtovat daně za telekomunikační služby či daň z prodeje, pokud to místní legislativa požaduje.

5 Závěr

Android studio se ukázalo být velice výkonným a praktickým nástrojem pro vývoj aplikací. Po pochopení jeho struktury a odlišností od klasického programování pro počítače se podařilo vytvořit požadovanou herní aplikaci pro chytré telefony. Hra byla testována na několika zařízeních s operačním systémem Android a díky dodržení zásad objektově orientovaného programování dopadly všechny testy úspěšně i navzdory rozdílným velikostem obrazovek. Největším problémem zde bylo porozumění hierarchii a funkcím jednotlivých částí Android studia. Syntaxe a použití objektového jazyka Javy se v prostředí Eclipse obešla bez potíží a většího časového zatížení. Při psaní kódu se podařilo demonstrovat všechny důležité prvky a možnosti vývoje, které lze využít při tvorbě dalších aplikací. Mnoho času a potíží ušetřilo promýšlení všech kroků dopředu, a proto se všechny předběžné návrhy ukázaly být velice užitečnými.

Nejlepším pomocníkem při vývoji byl samotný literární průzkum. Díky jeho zpracování, se podařilo porozumět základům práce s Android studiem a všem důležitým nástrojům pro tvorbu grafického rozhraní aplikace. Po jejím dokončení byla aplikace úspěšně umístěna na trh přes portál Google play.

Obrázek 18: Výsledná aplikace



6 Summary

Android studio has proven to be very useful and practical tool for application development. After understanding its structure and the differences from traditional computer programming a success came in the form of the finished game application for smartphones. The game was tested on several devices with the Android operating system and due to following the principles of object-oriented programming, it passed all the tests successfully even despite differences in screen sizes. The most serious problem here was to understand the hierarchy and functions of various parts of Android studio. The syntax and the use of object-oriented language Java in Eclipse was handled without difficulties and greater waste of time. The demonstration of all important features and development capabilities, that can be used to make other applications, was successful.

The most helpful tool for the development was the literary research itself. Thanks to its' processing, I managed to understand the basics of working in Android studio and all of the important tools used for creating the GUI of the application. After its' completion, the application has been successfully placed on the market through the Google Play portal.

Key words

Application, Android, Eclipse, smartphone, Java, programming

Seznam literatury

Knižní zdroje

[1] Liška, V. (2004). *Makroekonomie: druhé vydání.* Profesional Publishing, 628 s. ISBN 80-86419-54-1

[2] Eckel, B. (2000). *Myslíme v jazyku Java, knihovna programátora*. Grada Publishing, 432 s. ISBN 80-247-9010-6

[3] Kiszka, B. (2003). *1001 tipů a triků pro programování v jazyce Java*. Computer Press, 519 s. ISBN 80-7226-989-5

[4] Perry, P. J. (1996). *Java – tvorba dokonalých www stránek*. Grada Publishing, 328 s. ISBN 80-7169-415-0

[5] UBJÁNAJ, Miroslav. (2012) *Programujeme pro Android Vyd. 1.* Grada Publishing, 326 s. ISBN 978-80-247-3995-3

[6] MURPHY, Mark L., *Android 2: průvodce programováním mobilních aplikací.* Computer Press, 418 s. ISBN 978-80-251-3194-7

[7] BÁRTA, Jakub, (2016). *Vývoj aplikace v programovacím jazyce Java pro operační systém Android a její uvedení na trh.* Diplomová práce. České Budějovice: Jihočeská univerzita, Ekonomická fakulta. Vedoucí práce: Radim Remeš

Internetové zdroje

[8] Svetandroida, *kratke ohlednuti za historii androidu.* Vyvíjíme pro pro Android [online]. 2010 [cit. 5. 3. 2016]. Dostupné z: http://www.svetandroida.cz/

[9] KYPTA, Tomáš. *Vyvíjíme pro Android*. Vyvíjíme pro Android [online]. 2011 [cit 11. 3. 2016]. Dostupné z: http://www.svetandroida.cz/vyvijimeproandroid

[10] Android Developers [online]. 2014 [cit. 5. 3. 2016]. Dostupné z: https://developer.android.com/

[11] *Starting an Activity*. Android Developers [online]. 2014 [cit. 2. 3. 2016]. Dostupné z: https://stuff.mit.edu/afs/sipb/project/android

[12] *Get Started with Publishing*. Android Developers [online]. 2015 [cit. 28. 2. 2016]. Dostupné z: http://developer.android.com/distribute/googleplay

Seznam obrázků:

Seznam tabulek:

Seznam příloh

Příloha 1: Diagram tříd

Součástí diplomové práce je také CD obsahující:

- Elektronickou podobu textu ve formátu PDF
- Projektové složky se zdrojovým kódem cílové aplikace

Obrázek 19: Class diagram