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

Implementace hry pro OS Android

Bakalářská práce

Autor: Michal Macinka
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Hradec Králové srpen 2016

Prohlášení:

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

V Hradci Králové dne 26.8.2016

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Mgr. Tomáši Kozlovi, Ph.D. za metodické vedení práce. Mé poděkování patří také Ing. Janu Budinovi za cenné rady a věcné připomínky v rámci konzultací a při zpracování této práce.

Anotace

Cílem této bakalářské práce je zaměřit se na návrh a následnou implementaci hry pro zařízení na mobilní platformě Android pro více hráčů na lokální síti. Bakalářská práce je rozdělena do tří částí. První část se věnuje mobilní platformě Android, architektuře systému, komponentám aplikace a sestavovacímu systému této platformy. Druhá část se zabývá popisem moderních herních frameworků a enginů, které je možné použít pro vývoj her pro mobilní platformu Android. Součástí kapitoly o herních frameworcích a enginech je taktéž specifikace požadavků na výkonnosti cílové hry na platformě Android, měření výkonnosti jednotlivých technologií a stanovení nejvýhodnějšího řešení pro implementaci hry. Poslední část je zaměřena na návrh a vlastní implementaci vyvíjené hry.

Annotation

Title: Implementation of Game for Android OS

The aim of this Bachelor Thesis is to focus on the design and implementation of games for mobile devices on the Android platform for multiple players on a local network. This thesis is divided into three parts. The first part is dedicated to mobile Android platform, system architecture, application components and build system of this platform. The second part describes the modern game engines and frameworks that can be used to develop games for Android mobile platform. Part of this section is also contains a specification of requirements for the performance target game on the Android platform, measuring the performance of individual technologies and determine the best solution for the actual implementation of the game. The third part is focused on the design and implementation of developed game.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Mobilní zařízení se v dnešní době staly běžnou součástí každodenního života. Díky svému výkonu, různým hardwarovým vlastnostem a velkému množství senzorů dnes už často nahrazují fotoaparáty, GPS navigace, čtečky elektronických knih nebo MP3 přehrávače. Vzhledem k nepřebernému množství aplikací a her je možné používat mobilní zařízení i pro zábavu.

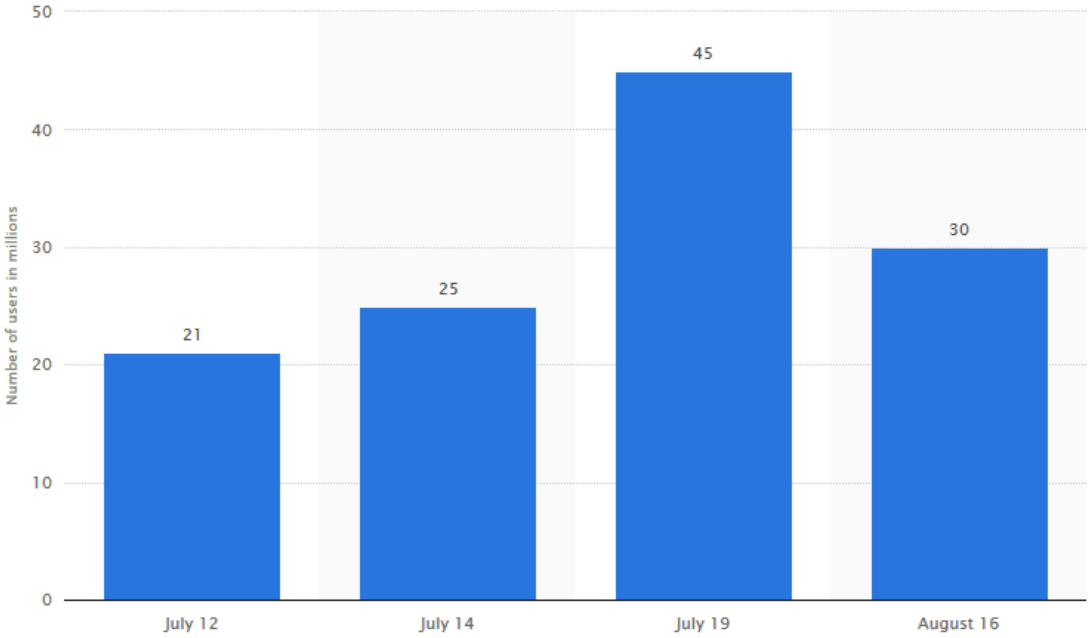
V Tabulka 1 je patrný nárůst chytrých telefonů mezi 2. čtvrtletím roků 2015 a 2016 o více než 14 milionů zařízení. Z tabulky je také zřejmý nárůst procentuálního podílu zařízení s operačním systémem Android a pokles počtu Apple iOS zařízení.

Operační systém	Počet zařízení v 2Q16	Podíl na trhu v 2Q16 (%)	Počet zařízení v 2Q15	Podíl na trhu v 2Q15 (%)
Android	296,912.8	86.2	271,647.0	82.2
iOS	44,395.0	12.9	48,085.5	14.6
Windows	1,971.0	0.6	8,198.2	2.5
Blackberry	400.4	0.1	1,153.2	0.3
Ostatní	680.6	0.2	1,229.0	0.4
Celkem	344,359.7	100.0	330,312.9	100.0

Tabulka 1 - Celosvětové prodeje chytrých telefonů v roce 2015 a 2016 (v jednotkách tisíců). Podle: [1]

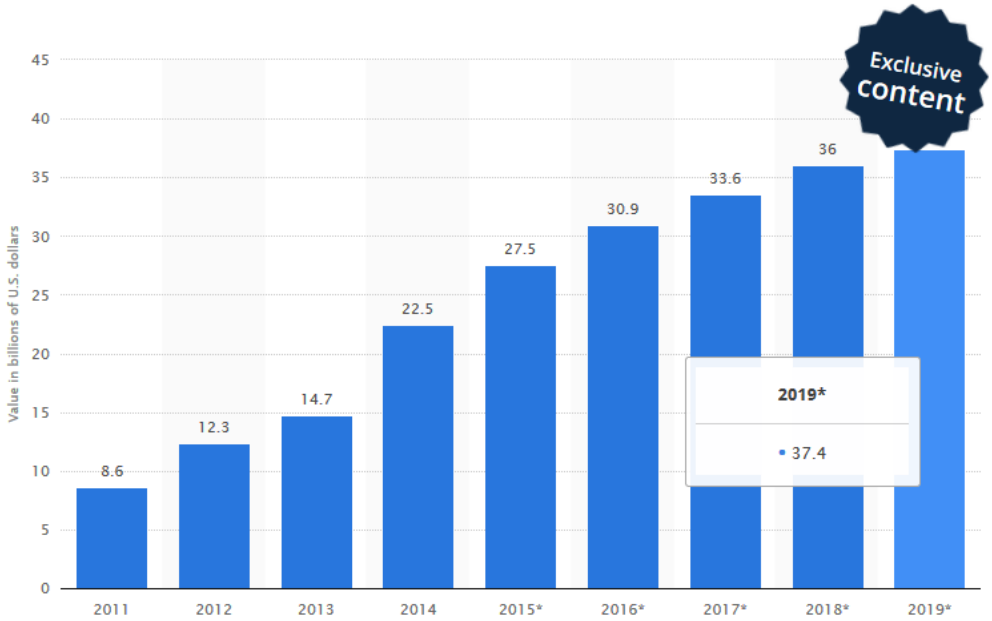
Díky výše uvedeným hodnotám vyplývá, že Android je platformou zajímavou pro budoucí vývoj aplikací a her už jen díky obrovské základně uživatelů.

Při vývoji je důležité rozhodnout se, kterým směrem se bude vývoj ubírat a jaká bude cílová skupina uživatelů. Zajímavou oblastí z hlediska vývoje se stává herní průmysl pro mobilní zařízení. Pro příklad je možné uvést mobilní hru Pokémon GO, která byla vydána na začátku července 2016. Krátce po vypuštění této hry překonaly počty uživatelů v USA hranici 45 milionů uživatelů. Na Obrázek 1 je zobrazen vývoj počtu uživatelů hry Pokémon GO v počátcích a po více než měsíci od vydání této hry.



Obrázek 1 - Vývoj počtu uživatelů krátce po uvedení a po více než měsíci od vydání hry Pokémon Go v milionech uživatelů. Podle: [2]

Za pomoci reklam a nákupu herního obsahu je možné mít určitý zisk z vytvořených her a aplikací. Mnoho herních vývojářů dává hráčům prostor nakoupit si herní obsah. Díky tomu může hráč být ve hře lepší než ostatní, ulehčí si postup hrou nebo jen získá možnost změnit si vzhled své postavy či zbraně v dané hře. Obrázek 2 zobrazuje postupný a očekávaný nárůst celosvětové hodnoty mobilního obsahu na trhu do roku 2019:



Obrázek 2 - Nárůst hodnoty mobilního obsahu na celosvětovém trhu od roku 2011 do 2019 (v milionech dolarů). Podle: [3]

2 Cíl práce

Cílem této bakalářské práce je seznámení s moderními herními frameworky a enginy pro vývoj her na mobilní platformě Android, zvolení nejvhodnější technologie dle herní specifikace a následně navrhnout a implementovat hru pro OS Android za pomoci vybrané varianty.

Pro výběru nejvhodnější technologie bude provedeno měření snímkové frekvence při různých složitostech herní scény pro tři odlišná zařízení s OS Android. V závěru měření proběhne sumarizace naměřených hodnot a zvolení nejvhodnější technologie pro implementaci ukázkové hry.

Jako výstup praktické části bakalářské práce bude vytvořena ukázková hra FirePlay postavená na technologii vybrané dle specifikace a výkonnosti daného herního frameworku nebo enginu.

3 Mobilní platforma Android

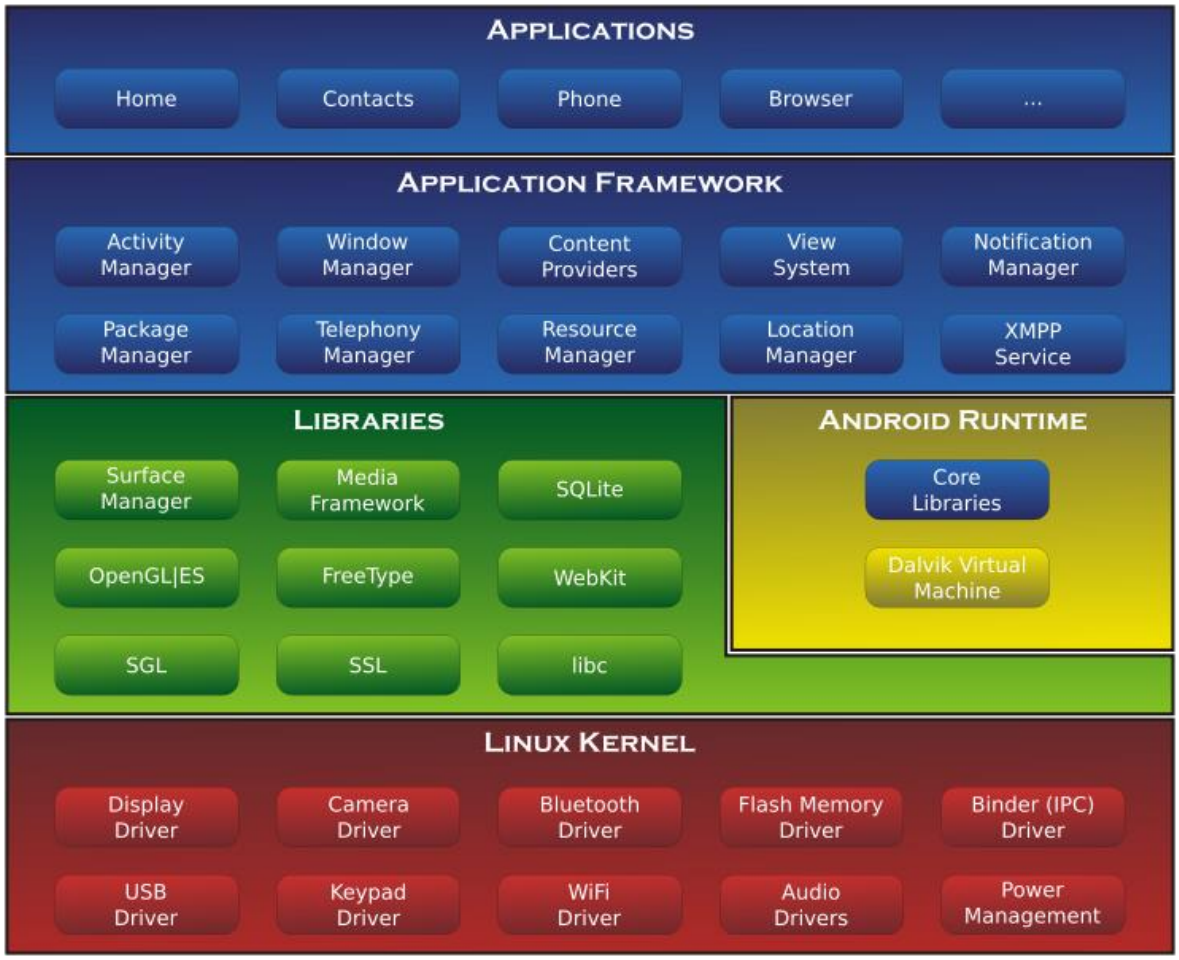
3.1 Operační systém Android

Android je mobilní operační systém založený na linuxovém jádru. Je vyvíjený společností Google a společenstvem Open Handset Alliance, které obsahuje 84 firem vyvíjejících otevřené standardy pro mobilní zařízení [4]. Operační systém Android je šířený jako open-source, to znamená, že vývojáři ani výrobci zařízení nemusí platit žádné licenční poplatky za vývoj pro tuto platformu. Operační systém je podle [5] licencován pod GNU licencí General Public Licence Version 2 (GPLv2), kde jakékoli změny od třetí strany musí splňovat open-source licenční podmínky. Aplikační framework je distribuován pod licencí Apache Software Licence (ASL/Apache2), která umožňuje distribuci obou verzí zdrojového kódu – otevřené i uzavřené. Vývojáři platformy mají možnost zlepšit Android bez nutnosti zveřejnit daná vylepšení open-source komunitě. Navíc z těchto zlepšení mohou profitovat a redistribuovat svou práci pod jakou licencí chtějí (například vylepšení specifická pro určitá zařízení). Stejně tak i vývojáři aplikací mají možnost šířit své aplikace pod licenčním schématem, který preferují. Mohou například vytvářet volně dostupné open-source aplikace, aplikace ze kterých mohou profitovat nebo jakoukoli variantu mezi tím.

3.2 Architektura operačního systému

Architektura operačního systému Android je složena z pěti vrstev [5]. Jednotlivé vrstvy provádí různé operace a jednají téměř samostatně, ale většinou dochází ke spolupráci jednotlivých částí.

Na následujícím obrázku jsou zobrazeny jednotlivé vrstvy operačního systému:



Obrázek 3 - Architektura operačního systému Android. Zdroj: [6]

Nejvyšší vrstvou jsou jednotlivé aplikace, které uživatelé používají. Ve středu architektury se nachází aplikační framework, knihovny a běhové prostředí. Nejnižší vrstvou architektury je linuxové jádro operačního systému neboli kernel.

3.2.1 Aplikace

Na vrcholu architektury jsou samotné aplikace, které jsou využívaný uživateli. Jedná se o stažené aplikace z některého z online obchodů, například Google Play či Amazon Store, nebo aplikace předinstalované v zařízení.

3.2.2 Aplikační framework

Vrstva, která je nejdůležitější a nejčastěji používána vývojáři. Open-source vývojová platforma operačního systému Android nabízí vývojářům rozmanité možnosti a prostředí pro vývoj komplexních a moderních aplikací. Aplikační framework programátorům

umožňuje přistupovat k různým službám, které mohou využívat přímo v aplikacích, například používat jednotlivé hardwarové prvky zařízení, spouštět ostatní aplikace na pozadí, využívat prvky uživatelského rozhraní a další.

3.2.3 Knihovny

Knihovny jsou další vrstvou této architektury. Pro vývoj aplikací existuje celá řada API rozhraní, která jsou pro všechna Android zařízení k dispozici. Základní API obsahují knihovny poskytující například přístup k základním službám operačního systému, vykreslování grafických prvků na obrazovku, nástroje pro zpracování a analýzu řetězců a jejich zobrazení, nízko-úrovňové třídy potřebné pro práci s kurzory v rámci databáze, několik standardních poskytovatelů obsahu pro běžné typy dat jako jsou audio a video soubory, obrázky, data v kalendáři atp., funkce pro práci s webovým obsahem a mnoho dalších.

Vedle těchto API existují knihovny, které jsou napsány v jazyku C/C++ a využívají různé systémové komponenty. Vývojářům jsou funkce naprogramovaných knihoven poskytnuty prostřednictvím aplikačního frameworku. Mezi tyto knihovny patří SGL a OpenGL/ES, které zajišťují podporu 2D a 3D grafiky, SQLite knihovna, která obsahuje odlehčenou verzi relační databáze pro mobilní zařízení a je dostupná všem aplikacím, knihovna SSL využívající pro bezpečnou internetovou komunikaci šifrovací protokol a další.

3.2.4 Běhové prostředí

Vrstva běhového prostředí obsahuje virtuální stroj DVM (Dalvik Virtual Machine) a základní knihovny jazyka Java. Od roku 2005 byl virtuální stroj Dalvik vyvíjen speciálně pro operační systém Android. Architektura Dalviku je registrově orientovaná a využívá základních vlastností linuxového kernelu jako je správa paměti, spolupráce běžících procesů nebo práce s vlákny. Tento virtuální stroj vznikl, protože programátoři vyvíjeli aplikace pro operační systém Android v programovacím jazyku Java. Knihovny jazyku Java mají licenci open-source, virtuální stroj JVM (Java Virtual Machine), který slouží pro kompilaci aplikací, ale volně šiřitelný není. Další důvodem bylo optimalizování výkonu a spotřeby energie virtuálního stroje pro potřeby mobilních zařízení.

Jak už bylo zmíněno, další součástí této vrstvy jsou knihovny programovacího jazyku Java. Obsah těchto knihoven je téměř srovnatelný s platformou Java SE (Standard Edition) s tím rozdílem, že API knihovny sloužící pro vytváření uživatelského rozhraní desktopových aplikací, byly nahrazeny knihovnami s obdobnou funkcí, ale pro operační systém Android. Dále přibyly knihovny pro práci se sítí.
Android Aplikace jsou vyvíjeny v jazyku Java, následně jsou kompilovány do Java byte kódu a nakonec pomocí Dalvik překladače do mezikódu. Výsledný byte kód je spuštěn na DVM. Každá aplikace je samostatným procesem s vlastní instancí DVM [7].

3.2.5 Jádro operačního systému

Jádro operačního systému představuje nejnižší vrstvu architektury. Základní funkcí jádra je úroveň abstrakce mezi použitým hardwarem zařízení, obsahuje všechny základní hardwarové ovladače jako fotoaparát, klávesnice, displej, atp., a softwarem ve vyšších vrstvách. Při startu zařízení je jádro zavedeno do operační paměti a je mu předáno řízení, což představuje neustálou kontrolu nad systémem a spolupráci všech běžících procesů, správu síťové komunikace, podpora správy pamětí atd.

3.3 Vývojové nástroje

3.3.1 Android SDK

Android SDK je sada nástrojů pro vývoj Android aplikací. SDK zahrnuje následující:

- potřebné knihovny
- debugger
- emulátor
- dokumentaci pro Android API
- jednotlivé verze platforem
- ukázky zdrojových kódů

Vždy když společnost Google vydá novou verzi operačního systému Android, vydá spolu s ní i korespondující Android SDK. Pro to, aby vývojáři mohli vytvářet aplikace s nejnovějšími funkcemi, musí si stáhnout a nainstalovat poslední verzi SDK.

3.3.2 Vývojové prostředí

Pro operační systém Android je možné využít různá vývojová prostředí (IDE = Integrated Development Enviroment). Společnost Google nabízí možnost využít oficiální vývojové prostředí Android Studio, které vychází z vývojového prostředí Intellij IDEA od společnosti JetBrains. Vedle této varianty je možné použít IDE, která jsou používána pro vývoj aplikací v programovacím jazyku Java, a to například Eclipse, NetBeans nebo už výše zmíněná Intellij IDEA.

3.4 Úroveň API a verze platformy

Úroveň API je unikátní celočíselná hodnota, která identifikuje verzi API frameworku nabídnutou verzí operačního systému Android. Platforma Android poskytuje API framework, který aplikace mohou využívat pro interakci s operačním systémem. API framework se skládá ze:

- základní sady balíčků a tříd
- sady XML elementů a atributů pro deklarování souboru s manifestem
- sady XML elementů a atributů pro deklarování a přístupu ke zdrojům
- sady Záměrů
- sady oprávnění, která aplikace může vyžadovat, stejně jako oprávnění zahrnutá v systému

Aktualizace API frameworku jsou navrženy tak, že nové API zůstává kompatibilní s předchozí verzí. To znamená, že většina změn v API jsou přídavné a představují novou nebo nahrazují starou funkcionalitu. Každá verze platformy Android podporuje právě jednu úroveň API, přestože je podpora implicitní pro všechny předchozí úrovně API (až do úrovně 1).

Při vývoji aplikací je důležité zvážit, která úroveň API bude použita. Android aplikace jsou zpravidla kompatibilní s budoucími verzemi platformy Android, protože převážná většina všech změn v API frameworku jsou změny přídavné. Aplikace vyvíjená pod specifickou verzí API (specifikovanou úrovní API) je kompatibilní s později vydanými verzemi platformy Android a vyšší úrovní API. Aplikace by měly být schopny běžet na všech

pozdějších verzích platformy Android, kromě případů kdy aplikace využívá část API, která byla z nějakého důvodu odebrána.

Budoucí kompatibilita aplikací je důležitá, protože spousta zařízení s OS Android dostává systémové aktualizace způsobem „over-the-air“ (OTA). Uživatel může nainstalovat aplikaci a úspěšně ji používat, poté dostane a nainstaluje OTA aktualizaci s novou verzí operačního systému. Jakmile je aktualizace nainstalována, aplikace poběží v nové běhové verzi prostředí.

V některých případech jsou změny v API takové, které jsou v systému samotném, tudíž mohou ovlivnit aplikaci, která běží v nové verzi běhového prostředí. Pro tyto případy je nutné, aby vývojář aplikace porozuměl, jak aplikace bude vypadat a jak se bude chovat v každém systémovém prostředí. Jako pomoc pro otestování aplikace na různých verzích platformy Android obsahuje Android SDK více verzí platforem, které je možné stáhnout. Každá platforma obsahuje kompatibilní obraz systému, který je možné spustit v AVD pro otestování aplikace.

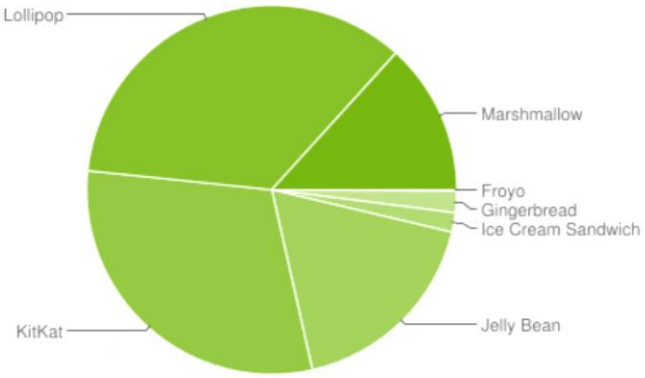
Při vývoji aplikace je nutné vybrat verzi platformy, oproti které se bude aplikace kompilovat. Obecně vzato by aplikace měla být kompilována oproti nejnižší verzi platformy, kterou aplikace bude podporovat [8].

Je možné určit nejnižší možnou verzi platformy úspěšnou kompilací aplikace vůči zařízením s touto verzí. Jakmile je určena nejnižší verze, je nutné, aby aplikace byla plně otestována na reálném nebo virtuálním zařízení, které bude používat korespondující verzi platformy (a úroveň API). Otestování by mělo proběhnout pro všechny platformy, které využívají vyšší úroveň API, než která je využívaná aplikací pro ověření budoucí kompatibility.

Pokud se sestavuje aplikace, která používá API nebo systémové vlastnosti, které byly uvedeny v poslední verzi platformy, minimální úroveň API by měla být nastavena na úroveň API této verze. Tím se zajistí, že uživatelé budou schopni aplikaci nainstalovat jen v případě, že jejich zařízení běží na kompatibilní verzi platformy Android. To také zajistí, že aplikace na jejich zařízení bude fungovat bez jakýchkoli potíží.

Následující obrázek zobrazuje tabulku a graf používaných verzí platformy, úrovně jejich API a jejich zastoupení mezi zařízeními s OS Android.

Version	Codename	API	Distribution
2.2	Froyo	8	0.1%
2.3.3 - 2.3.7	Gingerbread	10	2.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	1.9%
4.1.x	Jelly Bean	16	6.8%
4.2.x		17	9.4%
4.3		18	2.7%
4.4	KitKat	19	31.6%
5.0	Lollipop	21	15.4%
5.1		22	20.0%
6.0	Marshmallow	23	10.1%



Obrázek 4 - Procentuální zastoupení jednotlivých verzí Android OS mezi mobilními zařízeními. Zdroj: [38]

Při hodnotách uvedených v tabulce je možné kumulativním součtem jejich zastoupení mezi Android zařízeními stanovit verzi API, která by měla být použita pro aplikace s ohledem na jejich modernost. Minimální úroveň verze API v tomto případě připadá na API 15, což odpovídá verzi platformy 4.0.3-4.0.4 s označením Ice Cream Sandwich. Výsledek kumulativního součtu procentuálního zastoupení jednotlivých zařízení, která budou kompatibilní s tímto API, vychází na 97.8% všech zařízení s operačním systémem Android.

3.5 Komponenty aplikace

Základními stavebními bloky Android aplikací jsou komponenty. Komponenty jsou volně vázané na aplikační manifest AndroidManifest.xml. Aplikační manifest popisuje každou komponentu aplikace a její interakce s ostatními komponentami. V Android aplikacích je možné použít čtyři hlavní komponenty:

- Aktivity (Activities)
- Služby (Services)
- Přijímače (Broadcast Receivers)
- Poskytovatelé obsahu (Content Providers)

Vedle hlavních komponent existují komponenty, které se používají společně s hlavními pro vytvoření logiky a spojení mezi nimi. Mezi tyto další komponenty například patří:

- Fragmenty (Fragments) – reprezentuje část uživatelského rozhraní v Aktivitě
- Pohledy (Views) – elementy uživatelského rozhraní, které jsou vykresleny na obrazovce, například tlačítka, popisky, textová pole, atd.
- Rozložení (Layouts) – hierarchie, která ovládá formát obrazovky a vzhled jednotlivých elementů na ní
- Záměry (Intents) – zprávy, které propojují komponenty mezi sebou
- Zdroje (Resources) – vnější prvky jako např. řetězce, konstanty a obrázky
- Oznámení (Notifications) – komponenta, která se používá pro upozornění uživatele na určitou událost

3.5.1 Aktivity

Aktivita je komponenta aplikace, která představuje obrazovku, se kterou může uživatel provádět interakce v určitém pořadí pro dosažení určitého výsledku jako například pořídit fotografii, napsat email, prohlížet mapy atd. [5]. Každá aktivita je dána oknem, ve kterém je vykresleno uživatelské rozhraní. Okno zpravidla vyplňuje celou obrazovku, ale může být i menší než obrazovka a překrývat ostatní okna.

3.5.2 Služby

Služba představuje komponentu, která běží na pozadí a provádí dlouhotrvající operace. Služba může například přehrávat hudbu na pozadí, zatímco uživatel používá jinou aplikace nebo může stahovat data po síti bez toho, aniž by zablokovala uživatelské interakce s aktivitou.

3.5.3 Přijímače

Přijímače odpovídají na všesměrově vysílané zprávy z ostatních aplikací nebo systému. Pro příklad, aplikace zahájí vysílání, aby upozornila ostatní aplikace na to, že určitá data byla stažena a jsou dostupná k použití. Přijímač je ten, kdo zachytí tuto komunikaci a v závislosti na ní začne vykonávat konkrétní akce.

3.5.4 Poskytovatelé obsahu

Tato komponenta poskytuje data z jedné aplikace ostatním na vyžádání. Tyto žádosti jsou obslouženy metodami třídy ContentResolver. Data mohou být uložena v souborovém systému, SQLite databázi, na webu nebo na jiném persistentním úložišti ke kterému aplikace může přistupovat.

3.5.5 Záměry

Záměry jsou zprávy, které umožňují jednotlivým komponentám posílat žádosti aktivitám nebo ostatním komponentám. Záměry mají tři základní případy použití:

- spuštění aktivity
- spuštění služby
- doručení všesměrově vysílané zprávy

Existují dvě kategorie záměrů:

- explicitní – pro explicitní záměr se jednoznačně specifikuje cílová aplikace, která provede akci nad objektem
- implicitní – pro implicitní záměr není specifikovaná aplikace, takže při spuštění záměru je na výběru uživatele, kterou aplikaci pro vykonání akce vybere

3.5.6 Životní cyklus aktivity

Aplikace se obvykle skládá z několika aktivit, které spolu určitým způsobem souvisí. V aplikaci je typicky jedna hlavní aktivita, která je uživateli zobrazena po spuštění aplikace. Každá aktivita může spustit další aktivitu. Jakmile je nová aktivita spuštěna, předchozí aktivita je zastavena, systém tuto aktivitu zachová a uloží do zásobníku, tzv. „back stack“ [9]. Jakmile je nová aktivita spuštěna, je položena na vrch zásobníku a soustředí na sebe uživatelův focus. Back stack funguje na základním principu zásobníku „last in, first out“ (LIFO). Jakmile je uživatel dokončí interakci s aktuální aktivitou a stiskne tlačítko Zpět, aktivita vyskočí ze zásobníku, je zničena a běh aplikace pokračuje se přechozí aktivitě.

Na rozdíl od ostatních programovacích paradigmat, ve kterých jsou aplikace spouštěny hlavní metodou `main()`, systém Android inicializuje kód v instancích aktivit vyvoláním

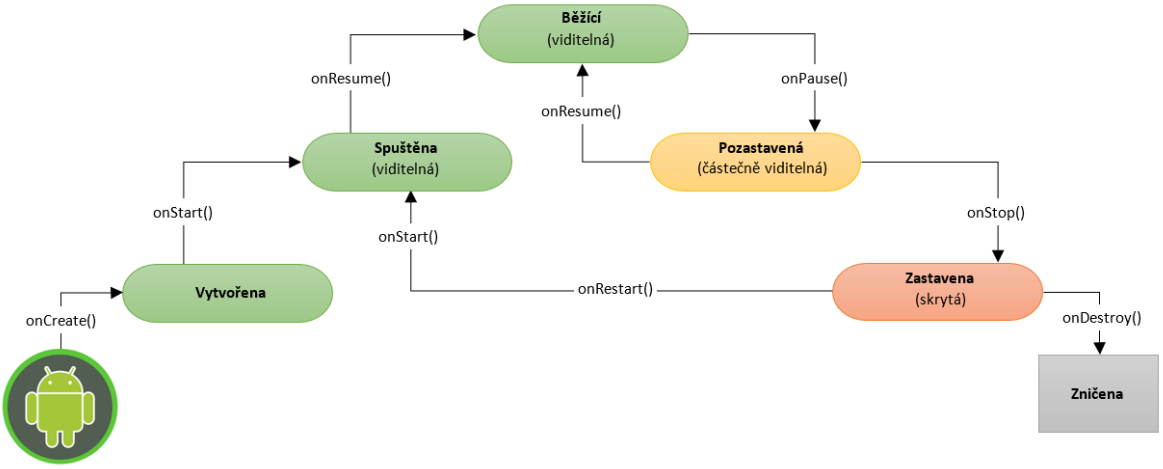
specifických metod zpětného volání, které korespondují konkrétním fázím životního cyklu aktivity. Jsou zde sekvence metod zpětného volání, které aktivity spouští a dále sekvence, které aktivity ukončují.

V průběhu života aktivity, systém volá základní sadu metod životního cyklu v sekvenci podobné pyramidě. Když systém vytvoří novou instanci aktivity, každé zpětné volání metody přesouvá aktivitu o krok k vrcholu. Vrchol pyramidy je bod, ve kterém aktivita běží v popředí a uživatel s ní může provádět interakce.

Každá aktivita podle [7] reaguje na následující metody:

- `onCreate()` – inicializační metoda, která je zavolána po spuštění aplikace a vytváří novou instanci aktivity s veškerým nastavení grafického rozhraní a datovými vazbami
- `onStart()` – tato metoda je zavolána v případě, že aktivita byla spuštěna poprvé, hned po metodě `onCreate()`, aktivita se stává viditelnou pro uživatele – spouští se tzv. viditelný životní cyklus aktivity
- `onResume()` – metoda, která je zavolána, když se aktivita dostane na vrchol zásobníku a je viditelná uživateli, který s ní může interagovat
- `onPause()` – zavolání této metody značí přechod aktivity na pozadí
- `onStop()` – metoda přesouvá aktivitu z částečně viditelného stavu na pozadí, při zachování veškeré otevřené hierarchie objektů
- `onRestart()` – metoda je zavolána, když aktivita přechází z pozadí do částečně viditelného stavu
- `onDestroy()` – tato metoda je opakem metody `onCreate()`, je volána před zrušením aktivity

Jakmile uživatel začne opouštět aktivitu, systém zavolá jinou metodu, která přesune stav aktivity v pyramidě dolů za účelem zničení aktivity. V některých případech se aktivita přesune v pyramidě dolů jen částečně a čeká (například když uživatel přepne do jiné aplikace) v bodu kdy bude aktivita opět přesunuta na vrchol (pokud se uživatel k aktivitě vrátí) a pokračuje z místa, kde ji uživatel opustil.



Obrázek 5 - Životní cyklus aktivity v Android OS. Podle zdroje: [36]

V závislosti na složitosti aktivity je pravděpodobné, že nebude třeba implementovat všechny metody životního cyklu. Nicméně je důležité porozumět všem a implementovat ty, které zajišťují to, že se aplikace chová tak jak uživatelé očekávají. Správná implementace metod životního cyklu aktivity zajišťuje, že aplikace funguje dobře v několika směrech zahrnující následující:

- nehavaruje v případě, že uživatel obdrží telefonní hovor nebo se přepne do jiné aplikace mezitím, co ji uživatel používá
- nespotřebovává cenné systémové prostředky, když ji uživatel aktivně nepoužívá
- neztrácí pokrok uživatele, pokud aplikaci opustí a později se k ní vrátí
- nehavaruje nebo neztrácí pokrok uživatele, v momentu, kdy obrazovka mění orientaci mezi orientací na šířku a výšku

Aktivita může v průběhu času nabývat třech stavů – běžící, pozastavená a zastavená.

- běžící – aktivita je v popředí a uživatel s ní může provádět interakce

- pozastavená – aktivita je částečně zakrytá jinou aktivitou – jiná aktivita, která je v popředí je částečně průhledná nebo nezakrývá celou obrazovku; pozastavená aktivita nepřijímá žádný uživatelský vstup a nemůže vykonávat žádný kód
- zastavená – aktivita je kompletně skryta na pozadí a není uživateli viditelná; zatím co je aktivita zastavena, instance aktivity a všechny stavové informace jako proměnné jsou zachované, ale nemůže vykonávat žádný kód

Ostatní stavy (vytvořena a spuštěna) jsou přechodné a systém rychle přechází z těchto stavů na další voláním metod životního cyklu. To je se děje hned potom, co systém zavolá metodu `onCreate()`. Po zavolání této metody následuje rychlé zavolání metody `onStart()`, která je okamžitě následována voláním metody `onResume()`.

3.6 Gradle

Sestavovací systém Androidu kompiluje aplikační zdroje a zdrojové kódy, zabalí je do APK balíčků, které se mohou testovat, nasazovat, podepsat a distribuovat. Android Studio využívá sestavovací systém Gradle, moderní sestavovací sadu nástrojů pro automatizaci a řízení procesu sestavení a zároveň umožňuje definovat flexibilní konfiguraci vlastních sestavení. Každá konfigurace sestavení může definovat svou vlastní sadu zdrojových kódů a zdrojů, zatímco využívá části společné pro všechny verze aplikace. Android Plugin pro Gradle spolupracuje se sestavovací sadou nástrojů pro zajištění procesů a konfigurovatelných nastavení, která jsou specifická pro sestavování a testování Aplikací pro Android.

Gradle a plugin pro Android běží nezávisle na Android Studiu. To znamená, že je možné sestavit aplikaci pro Android z Android Studia, příkazové řádky počítače nebo na počítačích, kde Android Studio není nainstalováno, jako jsou například servery s průběžnou integrací.

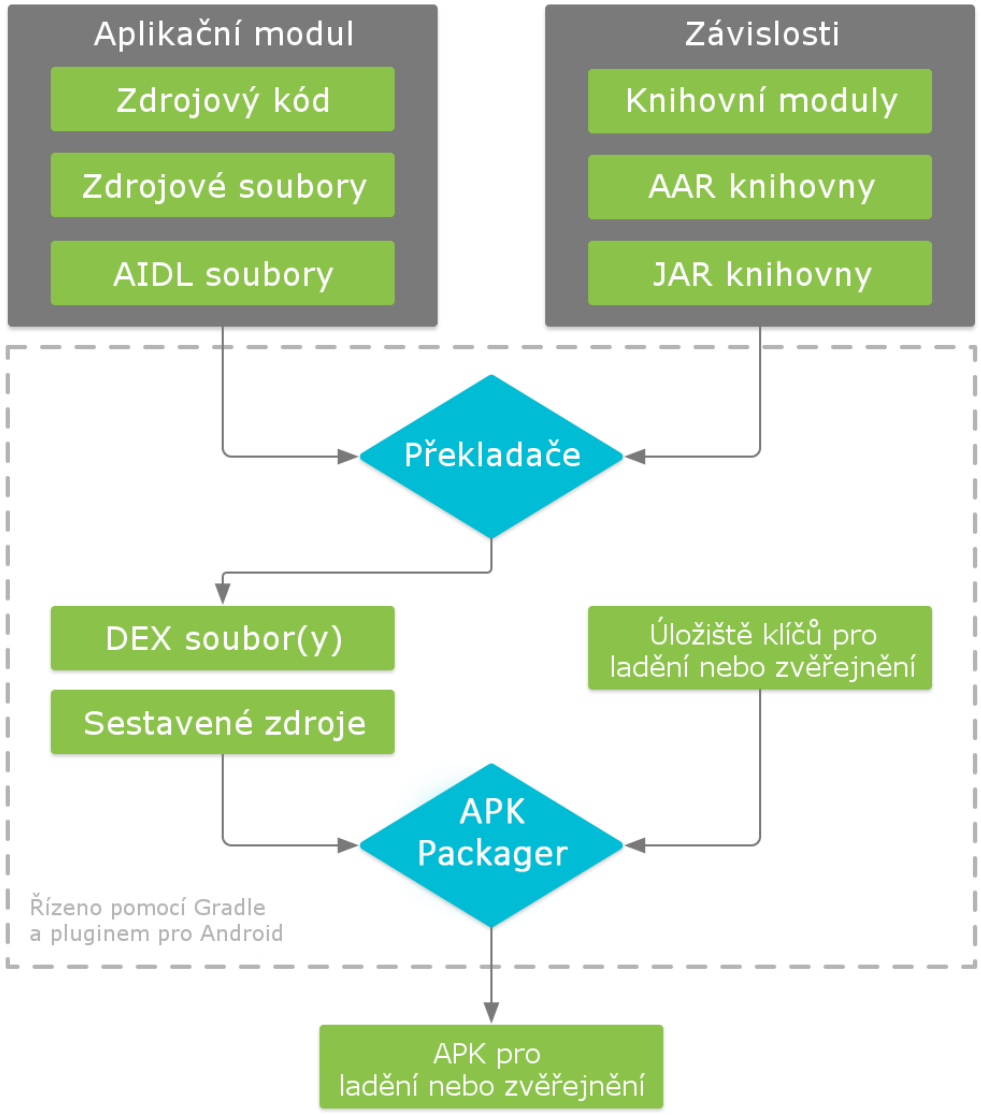
Flexibilita sestavovacího systému pro Android umožňuje vykonávat vlastní konfigurace sestavení, aniž se změnily zdrojové kódy aplikace.

Gradle je moderní varianta stále používaných sestavovacích systémů Maven nebo Ant, které bývaly dlouhou dobu sestavovacími systémy pro Android projekty. Oproti těmto sestavovacím systémům je Gradle mnohem snadnější na konfiguraci a tím je jednodušší ho používat.

Sestavovací skripty Gradlu nejsou psány tradičně ve formátu XML, ale doménově specifickém jazyku založeném na Groovy, dynamickém jazyku pro Java Virtual Machine [10].

3.6.1 Sestavovací proces

Sestavovací proces zahrnuje mnoho nástrojů a procesů, které přemění projekt v balíček aplikace pro Android, tzv. APK.



Obrázek 6 - Sestavovací proces Android aplikace. Podle zdroje: [11]

Sestavovací proces pro typický aplikační modul aplikace pro Android, jak je znázorněno na obrázku 4, se obecně skládá z následujících kroků:

1. Překladač přemění zdrojové kódy v DEX (Dalvik Executable) soubory, které obsahují byte kód – kód, který běží na Android zařízeních. Všechno ostatní je v zkompilovaných zdrojích.
2. Nástroj APK Packager zkombinuje DEX soubory se zkompilovanými zdroji do jediného APK souboru. Předtím než může být aplikace nainstalována a nasazena na zařízení s Androidem, musí být soubor APK digitálně podepsaný certifikátem.
3. APK Packager podepíše soubor APK buď pomocí úložiště klíčů pro ladění, nebo zveřejnění:
    a. V případě, že se vytváří ladící verze aplikace, která je pouze pro testování a profilování, APK Packager podepíše aplikaci s úložištěm klíčů pro ladění. Android Studio automaticky konfiguruje nové projekty s úložištěm klíčů pro ladění, které jsou podepsané samy-sebou a mají platnost 365 od podepsání.
    b. Pokud se vytváří verze aplikace, která je určena pro zveřejnění, APK Packager podepíše aplikaci s úložištěm klíčů zveřejnění. Tyto verze aplikací jsou podepsány soukromými certifikáty, které obsahují heslo, platnost a informace o autorovi, které se stanou součástí APK souboru.
4. Před vygenerováním finálního APK souboru, APK Packager použije nástroj pro optimalizování aplikace, aby používala méně operační paměti při běhu na cílových zařízeních.

Na konci sestavovacího procesu vznikne APK soubor aplikace pro ladění nebo zveřejnění, který je možný použít pro nasazení, testování nebo zveřejnění externím uživatelům.

3.6.2 Projekty a úlohy

V Gradle existují, mimo jiné, dva nejdůležitější koncepty – projekty a úlohy. Každé sestavení je tvořeno minimálně jedním projektem a každý projekt obsahuje jeden nebo více úloh. Každý soubor `build.gradle` reprezentuje projekt. Úlohy jsou jednoduše definované uvnitř sestavovacího skriptu. Jakmile se inicializuje sestavovací proces, Gradle shromáždí objekty typu projekt a úloha definované v sestavovacím souboru. Objekt úlohy se skládá ze seznamu objektů akcí, v pořadí ve kterém musí být vykonány. Objekty akcí jsou bloky kódu, které jsou vykonány podobně jako metody v jazyce Java.

3.6.3 Životní cyklus sestavení

Vykonávání sestavení pomocí Gradle je v nejjednodušší formě jen vykonání akcí v úlohách, které jsou závislé na ostatních úlohách. Pro zjednodušení sestavovacího procesu, sestavovací nástroje vytváří dynamický model pracovního procesu. To znamená, že všechny úlohy jsou zpracovány postupně. Výhoda postupného zpracování zabraňuje vytvoření smyček. Jakmile je jednou úloha vykonána, nebude už znovu volána. Úlohy bez závislostí vždy poběží dříve než úlohy se závislostmi. Graf závislostí je vygenerován v průběhu konfigurační fáze sestavení.
Sestavení pomocí systému Gradle má tři fáze [10]:

- Inicializace: V první fázi je vytvořena instance projektu. V případě že existuje více modulů, každý s vlastním souborem `build.gradle`, bude vytvořeno více projektů.
- Konfigurace: V této fázi jsou vykonány sestavovací skripty, vytváří a konfigurují se všechny úlohy pro každý projekt.
- Vykonání: Toto je fáze, kdy Gradle určí, které úlohy by měly být vykonány. Které úlohy by měly být vykonány, záleží na argumentech, které byly předány při spuštění sestavení.

4 Herní frameworky a enginy

4.1 Herní framework

Herní framework [12] je softwarová struktura, která slouží jako podpora při vývoji her. Zajišťuje abstrakci komunikace mezi hrou a operačním systémem. Skládá se z několika částí, které zajišťují určitý typ operací, jako je například vykreslování herní grafiky na obrazovku, přehrávání hudby a jiných zvuků, interakce s uživatelem, komunikace po síti nebo detekce kolizí. Při použití frameworku má vývojář plnou kontrolu nad kódem, tudíž si může uzpůsobit pracovní postup v projektu tak, jak uzná za vhodné.

4.1.1 libGDX

LibGDX je multiplatformní open-source vývojový framework, který je navržen převážně, ne však výhradně, na vytváření her pomocí programovacího jazyku Java. Kromě Javy je možné použít i programovací jazyk C nebo C++ pro začlenění jejich knihoven kvůli úlohám, které jsou závislé na výkonu. Jednou z výhod LibGDX je možnost spouštět a ladit kód přímo na desktopu. Toto umožňuje použít vlastnosti Java Virtual Machine (JVM), jako je například Code Hotswapping, která dovoluje okamžitě vidět výsledek změněného kódu přímo za běhu aplikace.

Důležitým poznatkem je, že LibGDX je framework, ne herní engine, se kterým často přichází spousta nástrojů jako například editory úrovní nebo kompletně předdefinovaný pracovní postup. Na jednu stranu se toto může zdát jako nevýhoda, ale ve skutečnosti to je jedna z výhod, která umožní si stanovit vlastní pracovní postup pro každý projekt. Například, LibGDX umožňuje pracovat na nízké úrovni, pokud je zapotřebí, je možné v určitém momentu zavolat funkce OpenGL ES (Embedded Systems). Nicméně, většinu času se pro realizování nápadů používá již zabudovaná funkcionalita LibGDX.

Platformy, pro které je možné vyvíjet [13]:

- Windows
- Linux
- Mac OS X
- Androidi OS 2.2 nebo novější
- iOS

- BlackBerry
- Java Applety (vyžaduje nainstalovaný JVM)
- HTML5 (pomocí JavaScript a WebGL)

Od prvního vydání LibGDX ve verzi 0.1 v březnu 2010 bylo na této knihovně odvedeno hodně práce. Poslední stabilní verze LibGDX je 1.9.3, která vyšla 16. května 2016 a obsahuje například následující vlastnosti [13]:

- Grafika
    - Vykreslování pomocí grafické knihovny OpenGL ES 2.0 na všech platformách
    - Nízko úrovňová komunikace s OpenGL
        - Textury, shadery, jednoduché vykreslování tvaru a další
    - Vysoko úrovňová 2D API
        - Ortogonální kamera, 2D částicový systém, bitmapová písma a další
    - Vysoko úrovňová 3D API
        - Perspektivní kamera, nahrávání 3D modelů, osvětlení, atd.
- Fyzikální a matematické výpočty
- Souborový systém
    - Abstrakce souborového systému pro všechny platformy
- Interakce s uživatelem
    - Abstrakce vstupních zařízení jako myši, dotykové obrazovky, klávesnice a různých senzorů
    - Detekce gest
- Nástroje pro práci s XML a JSON
- A mnoho dalších nástrojů a knihoven třetích stran

Pro vývoj aplikací v libGDX je zapotřebí následující [14]:

- Operační systém Windows, Mac OS X nebo Linux
- Java Development Kit 7+ (JDK)
- Android SDK
- Vývojové prostředí (IDE):
    - Android Studio

- Intellij IDEA
- Eclipse
- NetBeans

Vývoj pro operační systém iOS je možný pouze na platformě Mac OS X.

4.2 Herní engine

Herní engine [12] je systém určený pro vývoj her. Herní engine obvykle zajišťuje obdobné operace jako herní framework jako je například vykreslování herní grafiky na obrazovku, přehrávání hudby, detekce kolizí a jiné. Výhodou herních enginů je, že s nimi přichází spousta nástrojů jako editory úrovní a mnoho dalších. Nevýhodou v případě použití herního enginu může být, že vývojář nemá kontrolu nad zdrojovým kódem kvůli předepsanému pracovnímu postupu.

4.2.1 AndEngine

AndEngine je 2D OpenGL ES herní engine pro Android. Autorem enginu je Nicolas Gramlich. OpenGL je obvykle používáno pro 3D grafiku, ale umí velmi dobře pracovat i s 2D grafikou. AndEngine je herní engine, není to systém pro vytváření her. Převážná většina her pro operační systém Android je psaná v programovacím jazyku Java. To samé platí pro AndEngine. AndEngine je pouze knihovna, která se stará o určité aspekty hry.
Engine je open-source a je distribuovaný pod licencí Apache Licence Version 2.0. To znamená, že vytvářené hry mohou být volně stažitelné i komerčního typu a mohou být šířeny, pod jakou licencí vývojář uzná za vhodné. Licence, pod kterou je engine distribuován, umožňuje dělat zásahy do zdrojových kódů, ale tyto změny musí být zveřejněny pro veřejnost [15].
Pro vývoj v AndEnginu bylo dříve nutné přidat JAR (Java Archive) soubor do aplikace. Autor se rozhodl umístit celý engine do GitHub repositáře. Git je verzovací systém (VCS = version control system) pro zdrojové kódy projektů a GitHub je webová stránka pro Git repositáře. VCS nejen dovoluje ukládat zdroje projektů, navíc ukládá všechny verze jednotlivých souborů. Toto je užitečné v případě, že je potřeba historii změn nebo spolupracovat s více lidmi.
Nejpůsobivější aspekt AndEnginu je velmi snadné vytváření her. Možnost navrhovat a programovat hru během několika týdnů po prvním nahlédnutí do AndEnginu je možné,

ale to neznamená, že hra bude perfektní. Programování hry ale může být zdlouhavým úkolem v případě, že vývojář nerozumí tomu, jak engine funguje. Pro vývoj v AndEnginu je zapotřebí následující software a hardware [16]:

- Windows XP nebo novější, Linux nebo Mac OS 10.5.8 nebo novější operační systém
- Java SDK 7+
- Android SDK s balíkem nástrojů pro vývoj Android aplikací (ADT = Android Development Tools)
- emulované nebo reálné zařízení s operačním systémem Android 2.2 nebo novější

4.2.2 Corona SDK

Corona SDK je mobilní vývojová platforma, která umožňuje vytvářet 2D multiplatformní hry, aplikace a elektronické knihy. Tato vývojová platforma využívá pro vývoj skriptovací jazyk Lua, který je jednoduchý k pochopení a naučení. Hlavní výhodou tohoto enginu je, že není nutné mít rozsáhlé znalosti programování.
Nevýhodou je nutné připojení k Internetu pro sestavování aplikací. Tato nevýhoda je pouze v základní variantě Corona SDK. Corona SDK umožňuje vyvíjet aplikace pro platformy [17]:

- Android (včetně Android TV)
- iOS
- Kindle
- Windows Phone 8

Walter Luh, zakladatel společnosti Corona Labs, vytvořil Corona SDK v prosinci 2009. Corona Labs nabízí tři varianty jejich produktu – Corona SDK, Corona Enterprise Small Business a Corona Enterprise Unlimited. Varianty se liší ve funkcionalitě a v měsíčních poplatcích [18].
Corona SDK

- volně stažitelná varianta produktu

- přístup k více než tisíci API, zásuvným modulům (propojení se sociálními sítěmi Facebook a Twitter, integrace reklamy, …)
- žádné licenční poplatky
- přístup ke každodenním novým sestavením SDK
- Nevýhoda – online sestavení aplikací

Corona Enterprise Small Business

- Měsíční poplatek $79
- Obsahuje vše, co varianta Corona SDK
- Omezení v této variantě je limit příjmu do 500 tisíc dolarů
- Nativní knihovny a API – možnost volání C++ a Java knihoven
- Offline sestavení

Corona Enterprise Unlimited

- Měsíční poplatek $199
- Obsahuje vše, co přechozí varianty
- Bez limitu příjmu

Nejenže tato vývojová platforma umožňuje rychlejší způsob vývoje aplikací, je díky ní možné jednoduše zahrnout externí knihovny nebo systémy jako například [19]:

- Box2D - fyzikální knihovna, která simuluje fyzikální vlastnosti na různých objektech
- Google Play Game Services – umožňuje sdílení nejlepšího skóre a úspěchů ve hrách a jejich porovnávání s ostatními hráči v operačním systému Android
- Game Center – varianta Google Play Game Services pro operační systém iOS
- Sociální sítě – aplikace a hry je možné propojit se sociálními sítěmi jako je Facebook, Twitter, Google Plus a další
- Zpeněžení aplikací a her pomocí různých reklamních možností nebo nakupováním obsahu přímo v aplikacích

K vývoji aplikací je zapotřebí následující [20]:

- Pro vývoj na Mac OS X:
  - Operační systém Mac OS X 10.10 nebo novější
  - XCode 6.3 nebo novější
  - Textový editor například TextWrangler, BBEdit, TextMate nebo SublimeText Editor se zásuvným modulem Corona Editor

Na platformě Mac OS X lze vyvíjet aplikace pro operační systém iOS i Android. Výhodou platformy Mac OS X je možnost vývoje aplikací v grafickém editoru Composer GUI, vyvinutým společností Corona Labs.

- Pro vývoj na platformě Windows:
  - Operační systém Windows XP (Service Pack 3 a Internet Explorer 8) nebo novější
  - Alespoň 1GHz procesor
  - Grafickou kartu, která podporuje OpenGL 2.1
  - Textový editor například Notepad++ nebo SublimeText Editor se zásuvným modulem Corona Editor

Podpora zařízení a jejich operačních systémů je popsána v následující tabulce:

Platforma	Android OS	iOS
Windows	2.3.3 nebo novější (procesory ARMv7)	na platformě Windows není možné vytvářet aplikace pro operační systém iOS kvůli omezením společnosti Apple
OS X	2.3.3 nebo novější (procesory ARMv7)	6.0 nebo novější

Tabulka 2 - Podpora zařízení na vývojových platformách [20]

4.2.3 Cocos2d-x

Cocos2d-x je open-source herní engine používaný k vytváření her, aplikací a jiných multiplatformních interaktivních programů. Cocos2d-x umožňuje vývojářům využít jejich znalostí programovacího jazyka C++, skriptovacího jazyku Lua nebo JavaScript při vývoji multiplatformních aplikací pro různé platformy. Cocos2d-x je rychlý, jednoduchý k naučení a jeho použití šetří čas, úsilí vývojářů a cenu. Platformy, pro které můžeme v Cocos2d-x vyvíjet [21]:

- Android OS
- iOS
- Windows Phone
- Microsoft Windows
- Mac OS X
- Linux

Cocos2d-x je používán jak jednotlivci a nadšenci, tak velkými společnostmi. V dnešní době mnoho her vyvinutých pomocí Cocos2d-x dominují v žebříčcích nejvýdělečnějších aplikací na AppStore nebo Google Play hlavně v Číně, Jižní Koreji a Japonsku. Mnoho vývojářů z různých společností jako je například Google, Microsoft, Intel a mnoho dalších jsou aktivně zapojeni do komunity Cocos2d-x.
Hlavními rysy [22] Cocos2d-x aplikací je 2D grafika o kterou se stará grafická knihovna OpenGL 2.1 pro desktopové aplikace a OpenGL ES 2.0 pro mobilní aplikace, integrovaná fyzikální knihovna Chipmunk2D, správa obrazovky, hudba a zvukové efekty nebo síťová funkcionalita.
Minimální požadavky pro vývoj [23]:

- Mac OS X 10.7, XCode 5.1
- nebo Ubuntu 12.10, CMake 2.6
- nebo Windows 7, Visual Studio 2012
- Python 2.7.5
- Pro vytváření her pro Android OS je zapotřebí NDK r9d

Minimální požadavky pro spuštění [24]:

- Android OS 2.3+ pro Android hry
- iOS 5.0 pro iPhone/iPad hry
- Windows Phone 8 pro Windows Phone hry
- OS X 10.6 pro Mac hry
- Windows 7 pro Windows hry

Přehled podporovaných programovacích jazyků pro různé platformy:

	Platformy	C++	Lua	JavaScript
Mobilní	Android OS	ano	ano	ano
	iOS	ano	ano	ano
	Windows Phone 8	ano	ano	ne
Desktopové	Windows	ano	ano	ano
	OS X	ano	ano	ano

Tabulka 3 - Přehled programovacích jazyků a platforem v Cocos2d-x [25]

Pro vývoj v Cocos2d-x je možné využít rozsáhlou škálu editorů [26]. Oficiálními editory jsou

CocoStudio, oficiální editor světa, a Cocos Code IDE, oficiální integrované vývojové prostředí založení na vývojovém prostředí Eclipse. Dále můžeme využít různých editorů na vytváření úrovní, animací, textur a dalších.

4.3 Specifikace a požadavky

Výběr herního frameworku nebo enginu především závisí na požadavcích ve specifikaci implementované hry. Specifikace by měla mimo jiné obsahovat funkční a nefunkční požadavky.

Funkční požadavky popisují, co by implementovaná hra měla být schopna dělat. Mezi funkční požadavky se řadí například následující:

- způsob ovládání hry
- jak se hráč může pohybovat
- použití hudby a zvukových efektů ve hře
- v jakém rozlišení bude hra zobrazována
- počet jednotlivých úrovní hry a například časový limit jedné úrovně

Nefunkční požadavky specifikují, jakým způsobem má hra fungovat. Nefunkční požadavky jsou například:

- programovací jazyk jakým bude hra implementována – programovací jazyk může být často specifikován vývojovým jazykem pro cílovou platformu nebo framework/engine

- platforma nebo platformy, pro které bude hra cílena
- použitý framework nebo engine
- kompatibilita s verzemi specifikovaných platforem
- požadovaný výkon a spotřeba energie koncových zařízení
- podpora a použití knihoven a nástrojů třetích stran

Dalším ohledem pro zvážení při výběru by měly být i vedlejší body jako velikost a aktivita komunity, aktuálnost vybraného frameworku/enginu, zda vývoj frameworku/enginu probíhá nebo ne a další.

4.3.1 Požadavky

Dle specifikace uvedené v návrhu hry je výběr frameworku/enginu zohledněn čtyřmi nefunkčními požadavky: cílová platforma, kompatibilita, ohled na výkon koncových zařízení, podpora a použití knihoven a nástrojů třetích stran.
Cílovou platformou pro implementovanou hru byla vybrána platforma Android z důvodu převažující celosvětové rozšířenosti mezi používanými mobilními operačními systémy. Kompatibilita pro platformu Android byla stanovena při výběru úrovně API kumulativním součtem procentuálního zastoupení zařízení s operačním systémem Android na verzi platformy 4.0.3-4.0.4. Díky své otevřenosti a oblíbenosti mezi výrobci mobilních zařízení je Android operačním systémem široké škály mobilních zařízení. Vzhledem k minimálním požadavkům tohoto operačního systému dle [27] je nutné, aby hra byla schopna fungovat na velkém množství zařízení za optimálního výkonu. Posledním bodem je podpora a použití knihoven a nástrojů třetích stran. Jedná se o knihovny a nástroje například pro podporu síťové komunikace, zpracování 2D a 3D grafiky, editory úrovní, fyzikální a světelné knihovny a jiné.
Ačkoli ve specifikaci není explicitně uvedeno použití konkrétního programovacího jazyku, je nanejvýš vhodné použití programovacího jazyku Java vzhledem k použité platformě a počtu existujících knihoven, vytvořených právě za pomocí tohoto jazyku.
Při těchto specifikovaných požadavcích je možné pro vývoj hry použít Android SDK nebo jednu z možností:

- AndEngine
- libGDX

4.4 Testování výkonnosti

Ve filmové průmyslu je využívána snímková frekvence 24 Hz [28]. Hodnota 1 Hz se rovná 1 snímku za vteřinu (fps = frames per second). Tato frekvence se stala standardem pro filmový průmysl a je nejnižší možnou frekvencí, kdy pohyb pro lidský mozek působí souvislým dojmem. Při sledování natočeného videa při 24 fps se objevují mezery, které mozek automaticky doplňuje pomocí představivosti. Čím vyšší snímková frekvence je použita, tím méně toho mozek musí doplnit. V herním průmyslu, obzvlášť u akčních her z pohledu první osoby, záleží především na plynulosti herního obrazu. Na rozdíl od filmu, u her se dynamicky mění herní scéna. Počty objektů, grafické efekty a mnoho dalšího se mění dynamicky v čase a zároveň v závislosti na interakcích hráče, v případě online hry na více hráčích. U iOS/Android zařízení je maximální snímková frekvence obrazovky je 60 Hz. Zařízení nikdy nevykreslí více než 60 snímků za vteřinu. Na rozdíl od jiných platforem, neexistuje zde žádná možnost, jak z aplikace odemknout snímkovou frekvenci, aby mohla být zobrazována tak rychle, jak zařízení zvládne vykreslovat. Android a iOS zařízení používají vertikální synchronizaci pro aktualizaci obsahu obrazovky v pevném intervalu. To zabraňuje efektu trhání obrazu.

Pro implementaci ukázkové hry ze specifikovaných požadavků vyplývají tři varianty, které je možné pro vývoj použít: Android SDK, AndEngine a libGDX framework.

4.4.1 Postup testování výkonnosti

Pro výběr nejvhodnějšího nástroje byly vytvořeny jednoduché mobilní aplikace využívající každou ze zmíněných variant. Po spuštění aplikace je zinicializován logovací nástroj Logger [29] pro Android aplikace a vytvořena herní scéna, na kterou se vytvoří specifikovaný počet herních objektů. Specifikovaný počet objektů je přidán do kolekce pomocí cyklu. Každý objekt má v herní scéně danou pozici danou souřadnicemi x a y a směr pohybu po vytvoření daný opět souřadnicemi x a y. Počáteční hodnoty pozice objektů jsou v bodech [x, y] = [0, 0], směr pohybu je pro každý objekt náhodně generovaný za pomocí Java třídy Random [30]. Nástroj Logger do konzole vývojového prostředí zaznamenává každou vteřinu hodnoty snímkové frekvence obrazovky na Android zařízení.

Aplikace byla nainstalována pro tři zařízení s rozdílným hardwarovým vybavením. Hardwarové specifikace jednotlivých zařízení jsou následující:

1. **Samsung Galaxy S4** [31]
    - rozlišení: 1080x1920 pixelů
    - Chipset: Qualcomm APQ8064T Snapdragon 600
    - CPU: Quad-core 1.9 GHz Krait 300
    - GPU: Adreno 320
    - RAM: 2 GB
2. **Samsung Galaxy S3** [32]
    - rozlišení: 720x1280 pixelů
    - Chipset: Exynos 4412 Quad
    - CPU: Quad-core 1.4 GHz Cortex-A9
    - GPU: Mali-400MP4
    - RAM: 2 GB
3. **HTC Desire HD** [33]
    - rozlišení: 480x800 pixelů
    - Chipset: Qualcomm MSM8255 Snapdragon S2
    - CPU: 1.0 GHz Scorpion
    - GPU: Adreno 205
    - RAM: 768 MB

Testování výkonnosti spočívalo ve spuštění aplikace s předem daným počtem objektů na všech výše zmíněných zařízeních. Pro každé měření byl vybrán soubor prvních 25 hodnot zalogovaných výše uvedeným nástrojem Logger. Na základě naměřených hodnot byly stanoveny průměry a nejčastěji se vyskytující hodnoty, které jsou shrnuty v následující tabulce.

	Samsung Galaxy S4		Samsung Galaxy S3		HTC Desire HD	
Android SDK	**snímky za vteřinu (fps)**					
počet objektů	**průměr**	**modus**	**průměr**	**modus**	**průměr**	**modus**
50	60	60	60	60	42	42
100	59.8	60	59.6	60	29.9	30
300	56.4	56	52.3	52	22.6	23
500	40.9	41	34.8	35	16.4	16
1000	21.1	21	18.1	18	8.5	9
2500	8.7	9	7.2	7	3.5	4
5000	4.8	5	3.7	4	1.88	2
10000	2.1	2	2.2	2	1.011	1
AndEngine	**snímky za vteřinu (fps)**					
počet objektů	**průměr**	**modus**	**průměr**	**modus**	**průměr**	**modus**
50	60	60	60	60	41.3	41
100	60	60	60	60	40.1	40
300	56.9	57	56.9	57	39.8	40
500	56.6	57	54.1	54	34.1	34
1000	52.1	52	49.5	50	25.4	26
2500	49	49	46.4	47	23.9	24
5000	22.2	21	21	20	10.9	11
10000	12.1	12	11.5	12	6	6
libGDX	**snímky za vteřinu (fps)**					
počet objektů	**průměr**	**modus**	**průměr**	**modus**	**průměr**	**modus**
50	60	60	60	60	42	42
100	60	60	60	60	42	42
300	59.9	60	59.8	60	41.9	42
500	59.9	60	56.9	57	35.8	36
1000	59.7	60	56.8	57	35.9	36
2500	57.6	58	54.7	55	34.5	35
5000	37	38	35.2	34	22.2	21
10000	20.3	20	19.3	19	12.1	12

Tabulka 4 - Výsledky testování výkonnosti jednotlivých technologií

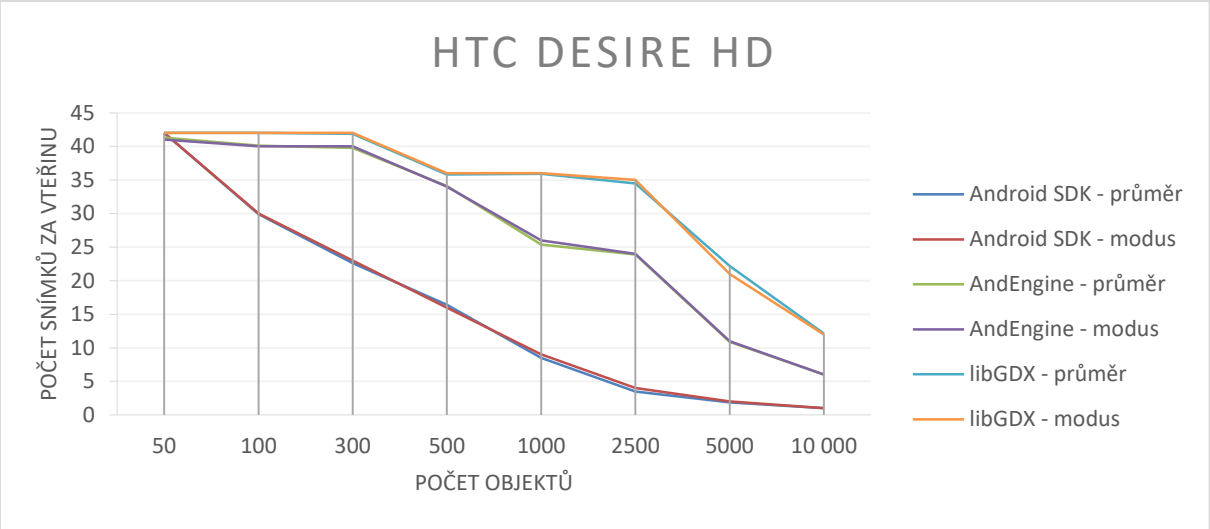
4.5 Sumarizace výsledků

Dle dosažených výsledků z měření výkonnosti jednotlivých technologií vyplývá, že výběr technologie pro implementaci mobilní hry závisí na počtu objektů vykreslených na herní scéně a zařízení, na kterém hra bude spouštěna. Hardware jednotlivých mobilních zařízení není možné vyměnit tak, jako je to možné u desktopových stanic. Jedním z hlavních cílů by proto měla být optimalizace hry tak, aby ji bylo možné spustit a hrát na široké škále zařízení při omezených hardwarových zdrojích.

Obecně platí, že čím vyšší je počet objektů k vykreslení na herní scéně, tím vyšší jsou požadavky na výkon zařízení. Toto tvrzení se potvrzuje za pomoci výsledků ve výše uvedené tabulce a dále z něj vyplývá, že čím vyšší je složitost scény, tím menší jsou hodnoty snímků za vteřinu.

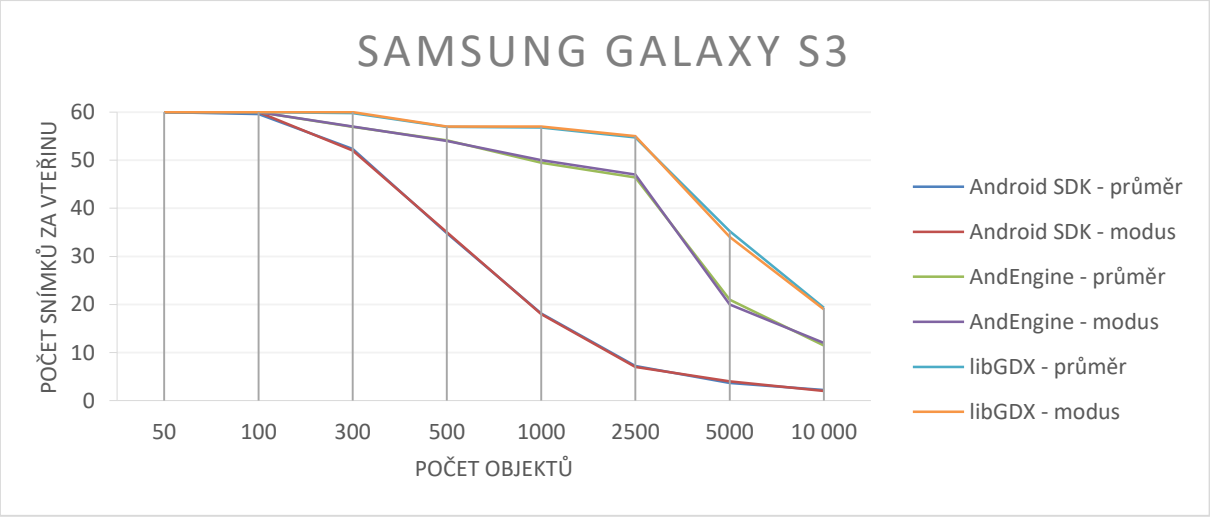
Dle výsledků je možné pro implementaci hry při nízkém počtu objektů použít libovolnou specifikovanou technologii. Toto tvrzení se v závislosti na vyšších počtech objektů mění. Samozřejmě ne vždy se stane, že ve hře se objeví tak velký počet objektů jako ten, který byl měřen při testování výkonnosti. Je ale možné, že bude nahrazen jinými grafickými efekty, jako jsou částicové efekty nebo různé animace, které ho mohou plně nahradit a tím si vyžádat vyšší výkon.

Pro Android SDK je očividné, že čistý kód v Javě může přinášet svobodu pro programátora, ale veškeré grafické operace pro vykreslení a optimalizaci musí řešit sám. Hodnoty uvedené v následujícím grafu svědčí například o tom, že u modelu HTC Desire HD při počtu objektů nad 300 nebude obraz plynulý, což je pro herní zážitek naprosto nevhodné.



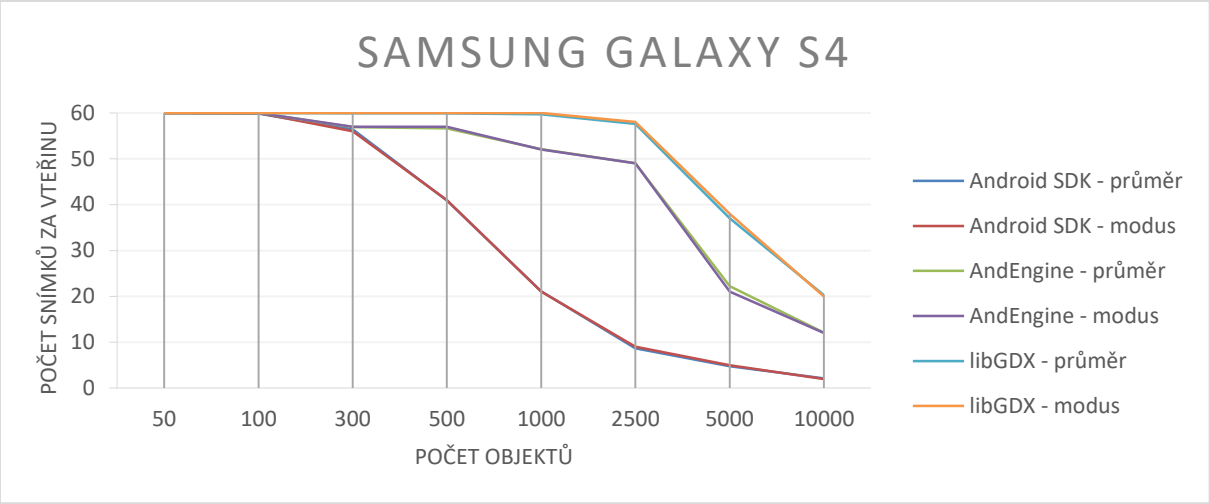
Obrázek 7 - Graf naměřených hodnot pro HTC Desire HD

Oproti testovanému zařízení HTC Desire HD je model Samsung Galaxy S3 lépe hardwarově vybaven. Toto potvrzují i naměřené hodnoty, které jsou shrnuty v grafu.



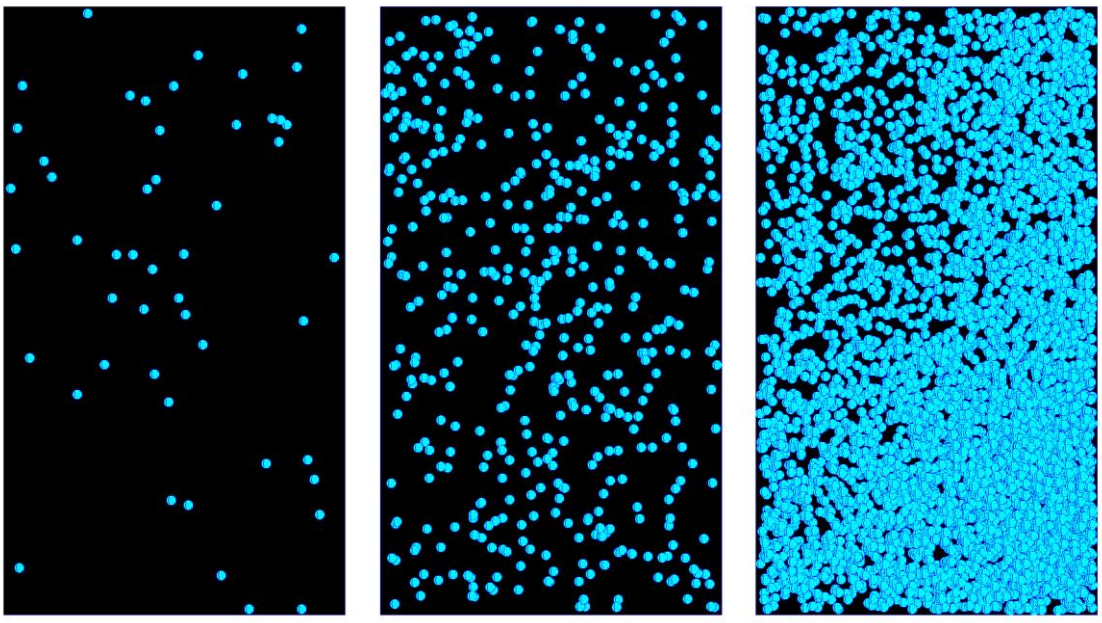
Obrázek 8 - Graf naměřených hodnot pro Samsung Galaxy S3

Model Galaxy S4 byl vydán s ročním odstupem od modelu Galaxy S3 jako jeho nástupce. Při porovnání hodnot v tabulce a jejich znázornění na jednotlivých grafech je zřejmé, že jejich hardwarové charakteristiky nejsou tolik odlišné například oproti testovanému zařízení značky HTC. Následující graf shrnuje naměřený hodnoty pro model Galaxy S4.



Obrázek 9 - Graf naměřených hodnot pro Samsung Galaxy S4

Jak vyplývá ze získaných výsledků, herní engine AndEngine a herní framework libGDX jsou oproti Android SDK schopné vykreslovat velké počty objektů při optimálním výkonu. Rozdíly jejich hodnot se téměř neliší ani při nejvyšších počtech objektů. Přesto, jak už bylo zmíněno, je vhodné, aby hra optimálně využívala výkon široké škály zařízení, high-endových modelů i modelů, které byly vypuštěny během posledních několika let. S ohledem na tento požadavek a přihlédnutí k výsledkům naměřených hodnot u modelu HTC Desire HD, který byl oficiálně uveden a vypuštěn dle [33] v roce 2010, je vhodnější použití libGDX frameworku, který mimo jiné zajistí řešení pro optimální grafický a výpočetní výkon.



Obrázek 10 - Ukázka testovací aplikace s 50, 500 a 5000 objekty.

5 Návrh hry

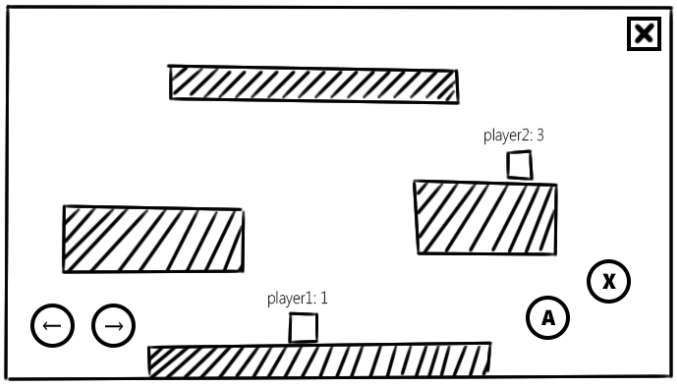
Při vývoji softwaru je důležitá specifikace vyvíjeného softwaru. Specifikace popisuje software a určuje funkční a nefunkční požadavky. Může také obsahovat případy užití, které popisují uživatelské interakce, které software musí poskytovat. Softwarová specifikace stanovuje dohodu mezi zákazníkem a dodavatelem a specifikuje, co softwarový produkt bude a nebude dělat. Měla by také poskytnout realistický základ pro odhadování výrobních nákladů, rizik, časového plánu a také zabraňuje neúspěchu softwarového projektu.

5.1 Herní specifikace

5.1.1 Koncept hry

Implementovaná hra s názvem FirePlay je 2D akční hra pro více hráčů. Herní scéna je hráči zobrazována ze strany, díky tomu má hráč přehled nad celou mapou. Hráči spolu v herní mapě soupeří a snaží se zasáhnout projektilem toho druhého nebo na protihráče skočit. V případě, že je jeden z hráčů zasažen, protihráč získá bod a zasažený hráč se objeví v místě, kde se objevil na začátku hry. Cílem hry je získat více bodů než protihráč do doby než se hráči rozhodnou hru ukončit.

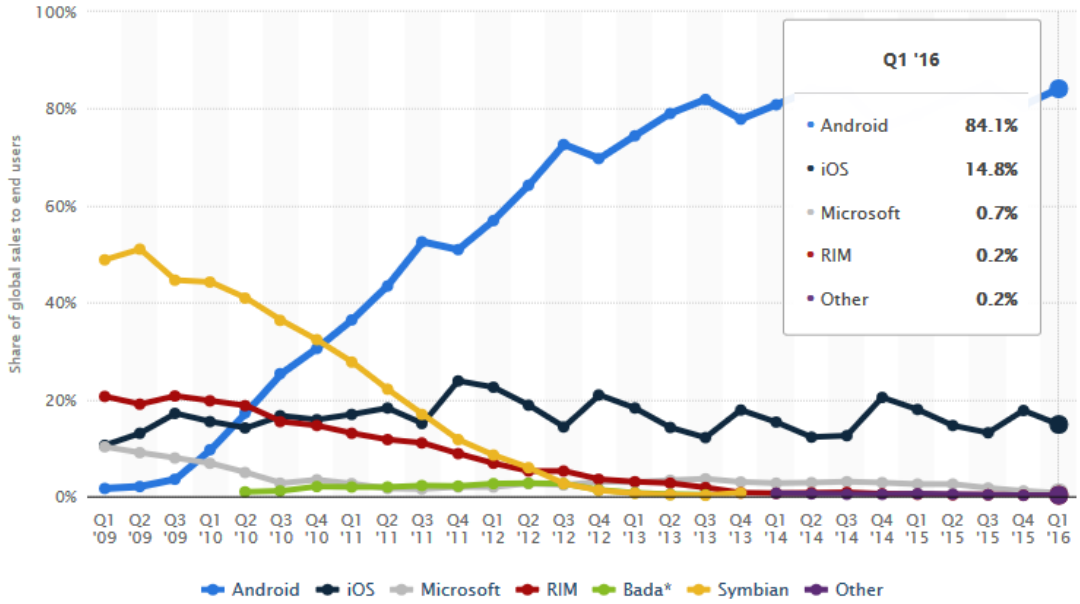
V základním konceptu, který představuje Obrázek 11, představují šrafované části objekty, po kterých se mohou hráči pohybovat. Hráčům je umožněn pohyb po herní scéně pomocí tlačítek doleva a doprava, na tlačítku A je implementována funkce skákání a na tlačítku X střelba. Postava hráče je v návrhu znázorněna bílými čtyřúhelníky, jménem hráče nad postavou a počtem získaných bodů za eliminaci protihráče. Ukončení hry je možné vykonat tlačítkem v pravém horním rohu.



Obrázek 11 - Základní koncept hry FirePlay

5.1.2 Cílová platforma

Cílovou platformou pro hru FirePlay byla vybrána platforma Android. Operační systém Android je mezi mobilními zařízeními jeden z nejrozšířenějších. Podle [34] v prvním kvartálu roku 2016 zastoupení operačního systému Androidi mezi všemi mobilními zařízeními je 84.1%.



Obrázek 12 - Procentuální zastoupení prodaných mobilních zařízení od roku 2009 do roku 2016 [34]

5.1.3 Požadavky

Požadavky implementované hry jsou následující:

- síťová komunikace na lokální síti pomocí Wi-Fi adaptéru v reálném čase
- 2D herní grafika
- optimální výkon pro širokou škálu mobilních zařízení
- kompatibilita knihoven a nástrojů třetích stran

6 Implementace hry

6.1 Síťová komunikace

Pro síťovou komunikaci mezi zařízeními byla použita knihovna KryoNet [35]. KryoNet je Java knihovna, která poskytuje čisté a jednoduché API pro efektivní TCP a UDP klient/server síťovou komunikaci pomocí NIO (Non-blocking I/O). NIO je kolekce Java API, které nabízejí funkce pro intenzivní I/O operace [36].

KryoNet používá serializační knihovnu Kryo pro automatický a efektivní přenos objektu grafů v síti. V objektově orientovaném programu, tvoří skupiny objektů síť prostřednictvím jejich vztahů mezi sebou navzájem a to buď přímým odkazem na jiný objekt, nebo řetězcem zprostředkujících referenci. Tyto skupiny objektů jsou označovány jako objekt grafů. Objektový graf je pohled na systém objektů v určitém časovém okamžiku. Běžný datový model jako je například UML diagram tříd zobrazuje detaily vztahů mezi jednotlivými třídami, objektový graf se zaměřuje na jejich instance. Diagramy objektů jsou podmnožiny celkového objektového grafu [37].

Pro komunikaci po síti je možné využít i jiné možnosti jako například Apache MINA, PyroNet nebo projekt Java Game Networking.

Důvodem pro vybrání této knihovny je její klient/server architektura a efektivita, což je obzvlášť důležité pro hry.

Při vytvoření hostované hry je zavolána metoda `hostGame()`:

```
private void hostGame() {
    Preferences preferences = Gdx.app.getPreferences("firePlay-
profile");
    GameScene gameScene = new GameScene (firePlay);
    gameScreen.hostGame();
    String name = preferences.getString("playerName");
    gameScene.loadLevel(map,"localhost",name);
    firePlay.setScene(gameScene);
}
```

- `Gdx.app.getPreferences("firePlay-profile")` – vrací odkaz na preference instance aplikace
- `new GameScene(firePlay)` – vytváří novou instanci herní obrazovky

- `gameScene.hostGame()` – spouští serverovou část aplikace na instanci herní obrazovky
- `preferences.getString("playerName")` – vrací jméno hráče uložené v preferencích, které při prvním spuštění aplikace nebo při změně jména hráč zadal
- `gameScene.loadLevel(map,"localhost",name)` – metoda přebírá jako vstupní parametry mapu, která bude nahrána pro herní scénu, hosta ke kterému se bude zařízení připojovat a jméno hráče
- `firePlay.setScene(gameScene)` – nastavuje herní obrazovku na právě vytvořenou herní scénu

Spuštění serverové části na instanci herní scény je následující:

```
public void hostGame() {
    isServer = true;
    server = new Server();
    NetworkRegisterer.register(server);
    server.start();
    try {
        server.bind(Constants.TCP_PORT, Constants.UDP_PORT);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

- `isServer = true` – hodnota při vytváření hostované lokální hry koresponduje s metodou pro nahrání herní scény, aby se odlišilo, zda se jedná o server nebo klienty
- `server = new Server()` – vytvoření nové instance serveru, pro obsluhu TCP nebo UDP připojení od mnoha klientů
- `NetworkRegisterer.register(server)` – metoda, která zaregistruje všechny třídy budou posílány po síti
- `server.start()` – spouští server na novém vlákně

- `server.bind(Constants.TCP_PORT, Constants.UDP_PORT)` – otevírá TCP a UDP server na daných portech, které pro hru FirePlay jsou: TCP – 4455 a UDP – 5544.

Po spuštění serveru na vlastním vlákně je nahrána herní scéna s výše specifikovanými vstupními argumenty.

```
public boolean loadLevel(String level, String host, String name)
{
    if (isServer) {
        world = new WorldManager(server);
    } else {
        client = new Client();
        NetworkRegisterer.register(client);
        client.start();
        try {
            client.connect(
                  Constants.TIMEOUT,
                  host,
                  Constants.TCP_PORT,
                  Constants.UDP_PORT);
        } catch (IOException e) {
            return false;
        }
    }

    renderer = new WorldRenderer(world, client, game);
    renderer.loadLevel(level, isServer, name);
    return true;
}
```

Jak bylo zmíněno výše, proměnná `isServer` koresponduje s metodou `loadLevel()`. Pomocí této proměnné se ověřuje, jestli je nahrávaná herní scéna na serveru nebo na klientovi. Při hostování lokální hry na serveru, bude vykreslen herní svět a přidán hostující hráč na jeho počáteční pozici. V případě, že by se jednalo o klienta je nejdříve zapotřebí vytvořit instanci klienta pro TCP nebo UDP připojení k serveru. Podobně jako u serveru je nutné zaregistrovat všechny třídy, které budou přenášeny po síti. Je důležité,

aby pro server i klienta byly třídy zaregistrovány ve stejném pořadí. Po zaregistrování bude klient spuštěn, stejně jako server, na vlastním vlákně. Klient se pokusí připojit na hostující server, který je definován IP adresou, kterou uživatel při stisku tlačítka „Join“ zadá. Při připojení je stejně jako u serveru vykreslen herní svět a hráči je přiřazena počáteční pozice, na které se bude objevovat.

O připojení hráče ke stávající hře se stará metoda `joinGame()`:

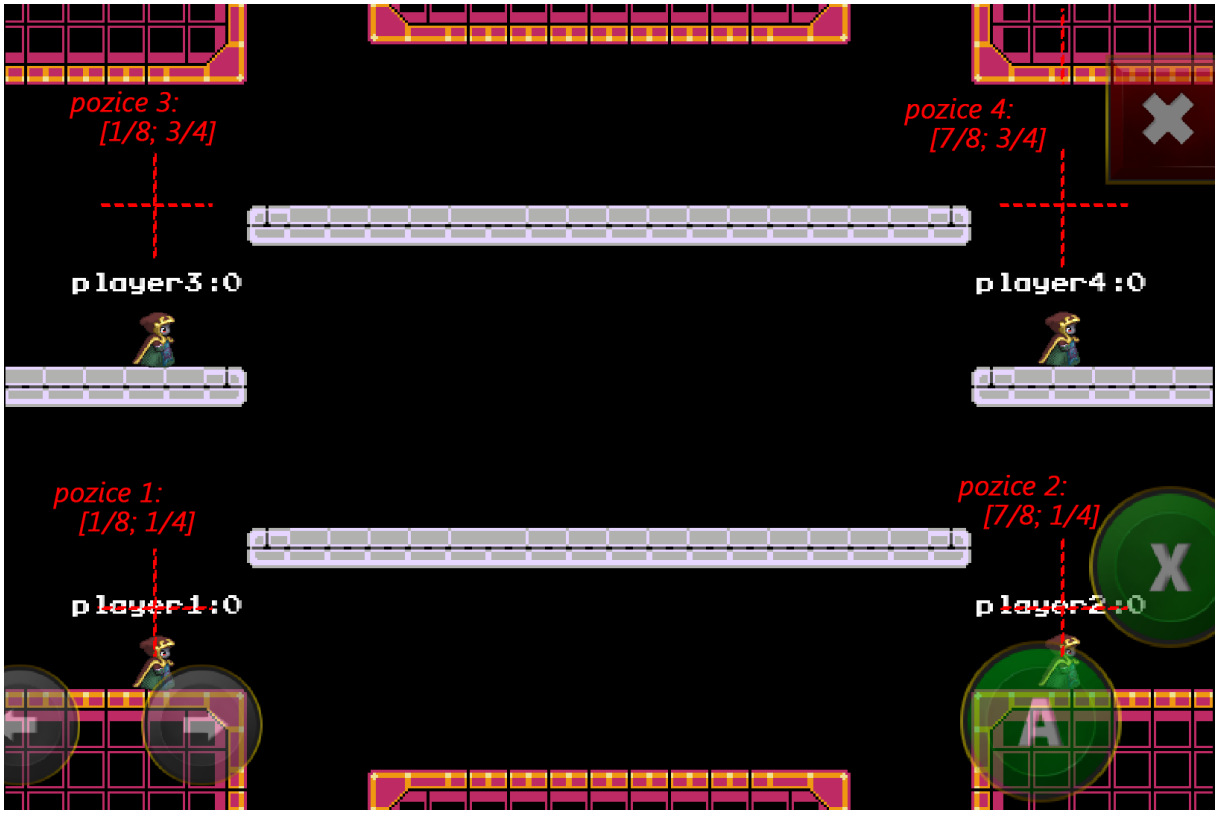
```
private void joinGame (final String host) {
        GameScene gameScene = new GameScene(firePlay);
        final Preferences preferences =
Gdx.app.getPreferences("firePlay-profile");
        String name = preferences.getString("playerName");
        if (gameScene.loadLevel(map, host, name)) {
                firePlay.setScene(gameScene);
        } else {
                gameScene.dispose();
        }
}
```

Průběh metody `joinGame()` je velmi podobný jako u metody `hostGame()`. Rozdíl je v nahrání herní úrovně. Herní úroveň je nahrána ze zařízení, které hostuje hru.

- `gameScene.loadLevel(map,host,name)` – metoda přebírá jako vstupní parametry mapu, která bude nahrána pro herní scénu, hosta ke kterému se bude zařízení připojovat a jméno hráče
- `gameScene.dispose()`- metoda `dispose()` je volána na instanci herní obrazovky, aby uvolnila všechny zdroje, které využívá

6.2 Vykreslování hráčů

V herním světě jsou definovány čtyři počáteční pozice, které jsou hráčům přiřazeny. Pozice jsou definovány vzhledem k velikosti výřezu, na kterém se odehrává hra. Z Obrázek 13 vyplývá, že počáteční pozice jednotlivých hráčů jsou na ose x v první a sedmé osmině šířky výřezu a první a třetí čtvrtině na ose y. Na následujícím obrázku jsou tyto pozice vyznačeny.



Obrázek 13 - Počáteční pozice pro vykreslení jednotlivých hráčů.

Ve zdrojovém kódu jsou pozice definovány následovně:

```
positions = new ArrayList<Vector2>();
positions.add(new Vector2(width*0.125f,height*0.25f));
positions.add(new Vector2(width*0.875f,height*0.25f));
positions.add(new Vector2(width*0.125f,height*0.75f));
positions.add(new Vector2(width*0.875f,height*0.75f));
```

- `new ArrayList<Vector2>()` – vytvoření nového seznamu s dvourozměrnými vektory
- `positions.add(new Vector2(…))` – vytvoření nového dvourozměrného vektoru a přidání do seznamu pozic
  - `width*0.125f,height*0.25f` – 1. pozice: x = 1/8 šířky výřezu a y = 1/4 šířky výřezu
  - `width*0.875f,height*0.25f` – 2. pozice: x = 7/8 šířky výřezu a y = 1/4 šířky výřezu
  - `width*0.125f,height*0.75f` – 3. pozice: x = 1/8 šířky výřezu a y = 3/4 šířky výřezu

- `width*0.875f,height*0.75f` – 4. pozice: x = 7/8 šířky výřezu a y = 3/4 šířky výřezu

O přiřazení těchto specifikovaných pozic jednotlivým hráčům, kteří se připojí k hostované hře se stará následující kód:

```
ServerPlayer player = new ServerPlayer(
            id++,
            positions.get(playerSpawnPoint).x,
            positions.get(playerSpawnPoint).y,
            bodyUtils);
            playerSpawnPoint++;
            playerSpawnPoint %= positions.size();
```

Při připojení nového hráče na server se vytvoří nová instance třídy `ServerPlayer` a je jí přiřazena pozice podle hodnoty proměnné `playerSpawnPoint`, která se s každým dalším hráče inkrementálně zvyšuje. Po každé inkrementaci se provádí dělení se zbytkem, kde je výsledek do proměnné přiřazen. Prvním čtyřem hráčům jsou postupně přiřazeny specifikované počáteční pozice. Příchod pátého hráče vynuluje proměnnou `playerSpawnPoint` a přiřazování pozic začíná opět od začátku.

6.3 Detekce kolizí

Detekci horizontálních a vertikálních kolizí jednotlivých objektů v ukázkové hře zajišťují metody `solveHorizontalCollision()` a `solveVerticalCollision()`. V případě kolize je nutné nastalou situaci adekvátním způsobem vyřešit a to například zamezením pohybu hráče skrze stěnu nebo podlahu.

6.3.1 Horizontální kolize

Mezi horizontální kolize patří především střetnutí hráče se stěnou, jiným hráčem nebo například projektilem jiného hráče. Následující Obrázek 14 znázorňuje příklad horizontálních kolizí mezi objekty v herním světě.

Obrázek 14 - Příklady horizontálních kolizí.

Metoda pro detekci horizontálních kolizí je následující:

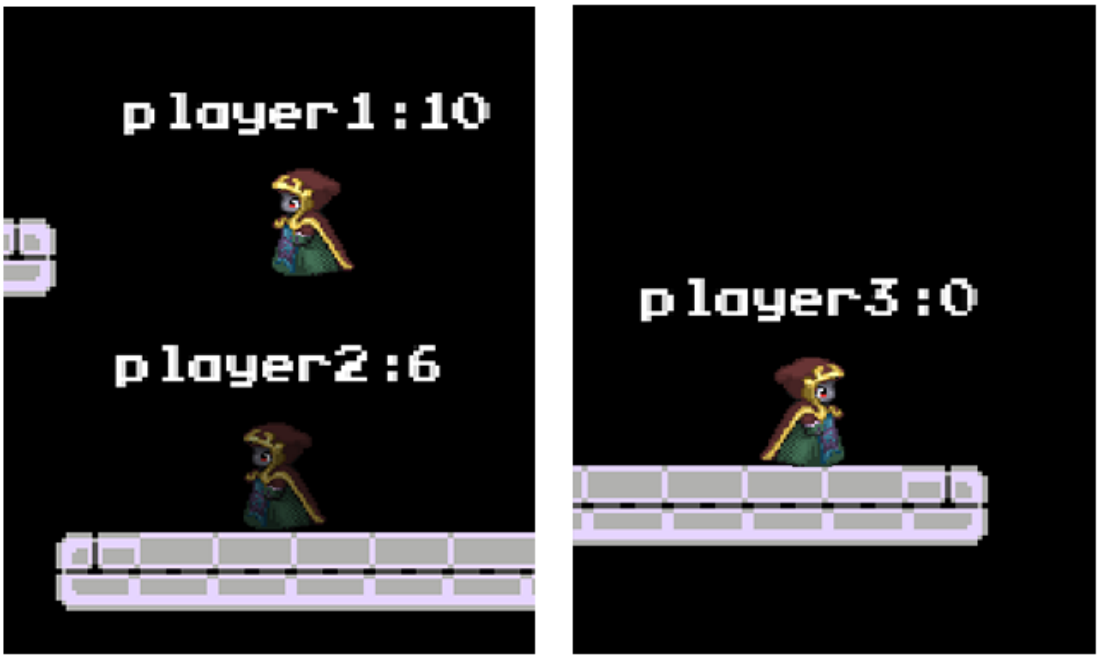
```
public void solveHorizontalCollision(Vector2 posVector, Body body) {
        if (velocity.x < 0) {
            posVector.x = body.rect.x + body.rect.width + 0.001f;
            velocity.x *= 0;
            CollisionUtils.solveLeftSideCollision(this, body);

        } else if(velocity.x > 0){
            posVector.x = body.rect.x - rect.width - 0.001f;
            velocity.x *= 0;
            CollisionUtils.solveRightSideCollision(this, body);
        }
        rect.setPosition(posVector.x, rect.y);
    }
```

Uvedená metoda `solveHorizontalCollision()` přijímá dva vstupní parametry: dvourozměrný vektor pro aktuální pozici objektu, například hráče, a tělo objektu se kterým proběhla kolize, například stěna. Uvedené podmínky zjišťují, zda je rychlost objektu kladná nebo záporná. V případě kladné hodnoty se objekt na herní scéně pohybuje doprava, při záporné doleva. Pozice pohyblivého objektu je stanovena součtem nebo rozdílem počáteční souřadnice stěny na ose x, její šířkou a hodnotou 0.001 zabraňující dosažení hraniční hodnoty stěny. Při kolizi objektů je rychlost pohyblivého objektu nulová.

Při horizontálních kolizích je také řešena kolize s projektilem protivníka.

6.3.2 Vertikální kolize

Mezi vertikální kolize se řadí střet hráče s plochou herního světa v podobě podlahy nebo s vrchní částí jiného hráče, například při skoku na hráče. Obrázek 15 zobrazuje příklady vertikálních kolizí v ukázkové hře.



Obrázek 15 – Příklad vertikální kolize při skoku na protivníka a stání na plošině ve hře FirePlay.

Vertikální kolize jsou v herním světě řešeny následujícím způsobem:

```
public void solveVerticalCollision(Vector2 posVector, Body body) {
        if (velocity.y > 0){
            posVector.y = body.rect.y - rect.height - 0.001f;
            velocity.y *= 0;
            grounded = false;
            CollisionUtils.solveJumpedOnCollision(this, body);
        } else if(velocity.y <= 0) {
            posVector.y = body.rect.y + body.rect.height + 0.001f;
            velocity.y *= 0;
            grounded = true;
            CollisionUtils.solveJumpOnCollision(this, body);
        }
        rect.setPosition(rect.x, posVector.y);
    }
```

Vertikální kolize jsou řešeny obdobným způsobem jako kolize horizontální, jen s rozdílem směru nahoru a dolu.
U vertikálních kolizí se objevuje proměnná `grounded`, která značí, zda je pohyblivý objekt na zemi. V případě, že objekt na zemi není, probíhá skok nebo pád z vyšší plošiny, není objektu umožněn další pohyb směrem nahoru.
Jak bylo zmíněno ve specifikaci, hráč může získat body dvěma způsoby. Zásahem soupeře projektilem nebo skokem na soupeře. O kolize s vrchní částí soupeřova těla se starají výše uvedené metody třídy `CollisionUtils`.

7 Závěr

Cílem této bakalářské práce bylo zvolení nejvhodnějšího moderního herního frameworku nebo enginu dle herní specifikace a následně navrhnout a implementovat hru pro operační systém Android pomocí této technologie.

Při výběru technologie bylo provedeno měření snímkové frekvence při různých složitostech herní scény pro tři odlišná zařízení s OS Android. V závěru měření proběhla sumarizace naměřených hodnot a zvolení nejvhodnější technologie pro implementaci ukázkové hry.

V rámci bakalářské práce byla vytvořena ukázková hra FirePlay postavena na frameworku libGDX. Při implementaci byly využity znalosti získané při studiu tohoto frameworku, kompatibilních knihoven a nástrojů třetích stran, které byly vyvinuty komunitou libGDX.

Mezi plánované budoucí rozšíření je přidání hry pro jednoho hráče s implementováním umělé inteligence pro entity ovládané počítačem, možnost hry více hráčů nejen na LAN síti nebo například nákupů v aplikaci a reklamy za účelem zisku.

8 Seznam použité literatury

[1] Android and Samsung lead in global smartphones sales during 2Q2016. *GadgetDetail* [online]. 2016 [cit. 2016-08-26]. Dostupné z: http://www.gadgetdetail.com/android-and-samsung-lead-in-global-smartphones-sales-during-2q2016/

[2] Number of daily Pokémon GO users in the United States as of August 2016 (in millions). *Statista* [online]. 2016 [cit. 2016-08-26]. Dostupné z: http://www.statista.com/statistics/589213/pokemon-go-user-number-us/

[3] Mobile contents market value worldwide from 2011 to 2019 (in billion U.S. dollars). *Statista* [online]. 2016 [cit. 2016-08-26]. Dostupné z: http://www.statista.com/statistics/292512/mobile-contents-market-value-worldwide/

[4] FAQ - Open Handset Alliance. *Open Handset Alliance* [online]. 2007 [cit. 2016-07-27]. Dostupné z: http://www.openhandsetalliance.com/oha_faq.html

[5] MACLEAN, Dave, Satya KOMATINENI a Grant ALLEN. *Pro Android 5*. [Fifth edition]. Berkeley: Apress, 2015. ISBN 1430246804.

[6] Android Architecture. *Embedded Linux Wiki* [online]. 2011 [cit. 2016-08-01]. Dostupné z: http://elinux.org/Android_Architecture

[7] ANNUZZI, Joseph, Lauren DARCEY a Shane CONDER. *Introduction to Android application development: Android essentials*. Fourth edition. Upper Saddle River, NJ: Addison-Wesley, 2014. ISBN 0321940261.

[8] SCHWARZ, Ronan., Phil DUTSON, James STEELE a Nelson TO. *The Android developer's cookbook: building applications with the Android SDK*. Second edition. Upper Saddle River, NJ: Addison-Wesley, 2013. ISBN 0321897536.

[9] Tasks and Back Stack. *Android Developers* [online]. b.r. [cit. 2016-08-01]. Dostupné z: https://developer.android.com/guide/components/tasks-and-back-stack.html

[10] PELGRIMS, Kevin. *Gradle for Android: Automate the build process for your Android projects with Gradle* [online]. Livery Place 35, Livery Street, Birmingham B3 2PB, UK: Packt Publishing, 2015 [cit. 2016-08-01]. ISBN 978-1-78398-682-8.

[11] The Build Process. *Android Developers* [online]. b.r. [cit. 2016-08-01]. Dostupné z: https://developer.android.com/studio/build/index.html

[12] MASOPUST, Ondřej. *Vývoj her pro mobilní platformu Android*. Brno, 2012. Bakalářská práce.

[13] LibGDX: Goals and Features. *LibGDX* [online]. libGDX, 2013 [cit. 2016-07-31]. Dostupné z: https://libgdx.badlogicgames.com/features.html

[14] NAIR, Suryakumar a Andreas OEHLKE. *Learning LibGDX Game Development*. 2. vydání. Livery Place 35, Livery Street, Birmingham B3 2PB, UK: Packt Publishing, 2015. ISBN 9781783554775.

[15] SCHROEDER, Jayme a Brian BROYLES. *AndEngine for Android game development cookbook: over 70 highly effective recipes with real-world examples to get to grips with the powerful capabilities of AndEngine and GLES 2*. Birmingham: Packt Pub., 2013.

[16] VARGA, Martin. *Learning AndEngine: Design and create Android games with the simple but powerful tool AndEngine*. Livery Place 35, Livery Street, Birmingham B3 2PB, UK: Packt Publishing, 2014. ISBN 978-1-78398-596-8.

[17] *Cross-Platform Mobile App Development for iOS, Android* [online]. 2015 [cit. 2016-07-08].

[18] Pricing. *Corona Labs* [online]. 2016 [cit. 2016-08-01]. Dostupné z: https://coronalabs.com/pricing/

[19] VALKOVIČ, Lukáš. *Porovnanie vývoja pre mobilné aplikácie v Corona SDK a libGDX*. Brno, 2014.

[20] System Requirements. *Corona Docs: System Requirements* [online]. 2016 [cit. 2016-07-08]. Dostupné z: https://docs.coronalabs.com/guide/start/systemReqs/index.html

[21] Developers Manual. *Cocos2d-x* [online]. b.r. [cit. 2016-07-08]. Dostupné z: http://cocos2d-x.org/wiki/Cocos2d-x

[22] Main Features. In: *Cocos2d-x* [online]. b.r. [cit. 2016-08-26]. Dostupné z: http://cocos2d-x.org/wiki/Cocos2d-x#Main-Features

[23] Build Requirements. In: *Cocos2d-x* [online]. b.r. [cit. 2016-08-26]. Dostupné z: http://www.cocos2d-x.org/wiki/Cocos2d-x#Build-Requirements

[24] Runtime Requirements. In: *Cocos2d-x* [online]. 2015 [cit. 2016-08-26]. Dostupné z: http://www.cocos2d-x.org/wiki/Cocos2d-x#Build-Requirements

[25] Supported Programming Languages. In: *Cocos2d-x* [online]. b.r. [cit. 2016-08-26]. Dostupné z: http://www.cocos2d-x.org/wiki/Cocos2d-x#Supported-Programming-Languages

[26] Supported Editors. In: *Cocos2d-x* [online]. b.r. [cit. 2016-07-31]. Dostupné z: http://www.cocos2d-x.org/wiki/Cocos2d-x#Supported-Programming-Languages

[27] Device Requirements. *Android Platform Development Kit* [online]. Google, 2008 [cit. 2016-07-31]. Dostupné z: http://www.netmite.com/android/mydroid/development/pdk/docs/system_requirements.html

[28] RUMSEY, Francis. a John WATKINSON. *Digital interface handbook*. 4rd ed. 70 Blanchard Road, Suite 402, Burlington: Focal Press, 2013. ISBN 0240519094.

[29] Class Logger. *Android Developers* [online]. b.r. [cit. 2016-07-31]. Dostupné z: https://developer.android.com/reference/java/util/logging/Logger.html

[30] Class Random. *Java SE Documentation* [online]. Oracle, c1993-2016 [cit. 2016-07-31]. Dostupné z: https://docs.oracle.com/javase/7/docs/api/java/util/Random.html

[31] Samsung I9505 Galaxy S4 Specification. *GSMArena* [online]. GSMArena, c2000-2016 [cit. 2016-07-31]. Dostupné z: http://www.gsmarena.com/samsung_i9505_galaxy_s4-5371.php

[32] Samsung I9305 Galaxy S III Specification. *GSMArena* [online]. GSMArena, c2000-2016 [cit. 2016-07-31]. Dostupné z: http://www.gsmarena.com/samsung_i9305_galaxy_s_iii-5001.php

[33] HTC Desire HD. *GSMArena* [online]. GSMArena, c2000-2016 [cit. 2016-07-31]. Dostupné z: http://www.gsmarena.com/htc_desire_hd-3468.php

[34] Global mobile OS market share in sales to end users from 1st quarter 2009 to 1st quarter 2016. *Statista* [online]. 2016 [cit. 2016-08-14]. Dostupné z:

http://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

[35] KryoNet library. In: *GitHub* [online]. 2016 [cit. 2016-08-15]. Dostupné z: https://github.com/EsotericSoftware/kryonet

[36] Java™ I/O, NIO, and NIO.2. *Java SE 7 I/O-related APIs - Developer Guides* [online]. c1993-2016 [cit. 2016-08-15]. Dostupné z: http://docs.oracle.com/javase/7/docs/technotes/guides/io/

[37] SPIEGEL, Andre. Object graph analysis. In: *Dissertationen online der Freien Universität Berlin* [online]. 1999 [cit. 2016-08-15]. Dostupné z: http://www.diss.fu-berlin.de/docs/servlets/MCRFileNodeServlet/FUDOCS_derivate_000000000347/tr-b-99-11.pdf

[38] Platform Versions. In: *Android Developers* [online]. b.r. [cit. 2016-07-27]. Dostupné z: https://developer.android.com/about/dashboards/index.html

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2015/2016

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

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Macinka Michal	Tovární 358, Žamberk	I1301031

TÉMA ČESKY:

Implementace hry pro OS Android

TÉMA ANGLICKY:

Implementation of Game for Android OS

VEDOUCÍ PRÁCE:

doc. Mgr. Tomáš Kozel, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl práce:
Popsat problematiku tvorby her pro mobilní platformu Android a navrhnout a implementovat ukázkovou hru pro tuto platformu.

Osnova:
1. Úvod
2. Mobilní platforma Android
3. Herní frameworky
4. Návrh hry
5. Implementace hry
6. Závěr

SEZNAM DOPORUČENÉ LITERATURY:

S. Nair and A. Oehlke, Learning LibGDX Game Development: Wield the power of the LibGDX framework to create a cross-platform game, 2. vydání.. Livery Place 35, Livery Street, Birmingham B3 2PB, UK: Packt Publishing, 2015.

Podpis studenta: Macinka **Datum:** 14.1.2015

Podpis vedoucího práce: **Datum:** 14.1.2015