VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HRA S PRVKY ROZŠÍŘENÉ REALITY PRO PLATFORMU ANDROID

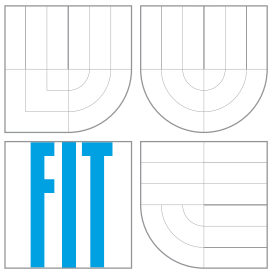
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROBERT PÖSEL

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

HRA S PRVKY ROZŠÍŘENÉ REALITY PRO PLATFORMU ANDROID

AUGMENTED REALITY GAME FOR ANDROID PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — ROBERT PÖSEL
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá návrhem a implementací frameworku pro geolokační hry na platformě Android. Jeho použití je následně demonstrováno na ukázkové aplikaci. V tomto textu jsou také popsány některé geolokační hry a možnosti vytváření 3D grafiky, zpracování senzorových dat a využití prvků rozšířené reality na platforně Android.

Abstract

The aim of this thesis is to design and implement framework for geolocation games on Android platform. Its use is then demonstrated on sample application. In this text are also described some geolocation games and ways of creating 3D graphics, processing data from sensors and use of augmented reality elements on Android platform.

Klíčová slova

Android, geolokační hry, herní framework, mobilní aplikace

Keywords

Android, geolocation games, game framework, mobile applications

Citace

Robert Pösel: Hra s prvky rozšířené reality pro platformu Android, bakalářská práce, Brno, FIT VUT v Brně, 2013

Hra s prvky rozšířené reality pro platformu Android

Prohlášení

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

.....
Robert Pösel
14. května 2013

Poděkování

Rád bych zde poděkoval vedoucímu práce, Ing. Aleši Láníkovi, za jeho konstruktivní připomínky, ochotu a energii, kterou mi při tvorbě této práce věnoval. Také děkuji mému světu, který se o mě stará tak dobře, že je vždy všechno přesně tak, jak má být.

Obsah

Kapitola 1

Úvod

Spolu s technickým vývojem mobilních zařízení se postupně vyvíjí i jejich programové vybavení a způsob, jakým lidé tyto technologie využívají. Mobilní telefony už zdaleka nejsou určeny jen pro telefonování nebo psaní krátkých zpráv. V dnešní době jsou to univerzální zařízení, které dokáží vykonávat mnoho různých činností, pro které byly v minulosti potřeba specializované nástroje. Fotoaparáty, přehrávače hudby a videa, navigační systémy, herní konzole a do určité míry dokonce i stolní počítače mohou být úspěšně nahrazeny chytrým telefonem či tabletem. Hlavní výhodou těchto zařízení není jen jejich vysoký výkon (často srovnatelný s jen o několik let staršími počítači) v kombinaci s jejich nízkou hmotností a velikostí (takže nám mohou být kdekoliv k dispozici), ale jsou to právě možnosti, které z této kompaktnosti a mobility vycházejí.

Většina telefonů má integrovaný fotoaparát, GPS modul, senzory pohybu a další. To umožňuje jejich využití v různých odvětvích běžného života, mezi něž patří i oblast zábavy a her. Propojení virtuálního a reálného světa je nyní jednodušší a aplikace či hry s prvky rozšířené reality se stávají běžnou záležitostí – ať už se jedná o vyobrazení herního světa do toho reálného skrze obraz z kamery, zpracování dat polohových senzorů pro práci se scénou nebo třeba využití přibližné polohy v geolokačních hrách.

A právě geolokační hry se dnes těší velké oblibě. Internet a sociální sítě pomáhají sbližovat lidi natolik, že už jim kontakt s pouhým počítačem častokrát nepřijde tak zajímavý jako dříve. Nyní se na trhu objevují různé možnosti, které lidem zpřístupňují spojení virtuální zábavy s jejich reálnými životy. Množství nástrojů, které se dají využít k tvorbě tohoto typu aplikací však zatím není velké. Cílem této práce je proto vytvoření herního enginu pro geolokační hry, který by usnadnil jejich tvorbu a tím pádem pomohl i jejich rozšíření.

V tomto textu se nejprve seznámíme s tím, co to vlastně geolokační hry jsou, rozdělíme si je do několika skupin a představíme si některé konkrétní příklady. V další kapitole se budeme zabývat vývojem na platformě Android. Zjistíme, jak se zde pracuje s 3D grafikou, jaké hardwarové senzory můžeme využít a jak s nimi dále pracovat. Na konci si povíme něco o tvorbě aplikací s prvky rozšířené reality a nástrojích, které nám mohou tuto práci zjednodušit. Čtvrtá kapitola pojednává o návrhu vlastního frameworku, jehož cílem je usnadnění tvorby geolokačních her a zapouzdření často používaných funkcí. V páté kapitole je uveden popis implementace této knihovny, jaké technologie byly využity, jakým způsobem je uložen herní scénář a jak se načítá, jak pracuje hlavní herní smyčka a v neposlední řadě i to, jak framework prakticky použít. Také je zde uvedeno, jakým způsobem probíhalo testování. Práci uzavírá kapitola se závěrečným zhodnocením projektu a možnostmi jeho pokračování.

Kapitola 2

Geolokační hry

Pojmem *geolokace* označujeme metodu, která pomocí různých technik umožňuje zjištění polohy konkrétního objektu [18]. V tomto textu budeme pojem geolokace chápat nejčastěji ve významu určení polohy mobilního zařízení, resp. uživatele totoho zařízení. Geolokační hry (anglicky *Location based games*) jsou tedy hry, které nějakým způsobem závisí na aktuální poloze hráče (hráčů). V této práci se zaměříme pouze na *mobilní geolokační hry*, k jejichž hraní je potřeba mobilní zařízení jako chytrý telefon či tablet.

2.1 Rozdělení mobilních geolokačních her

Tyto hry můžeme rozdělit do několika skupin různými způsoby: dle toho, jaké prostředky jsou ke hře potřeba (GPS, připravené herní prostředí, mobilní zařízení), dle délky jedné hry nebo třeba podle počtu hráčů.

2.1.1 Rozdělení dle délky trvání jedné hry

Délku trvání jedné hry budeme chápat jako čas potřebný ke splnění (či nesplnění) cíle hry, což následně vede k jejímu ukončení. Můžeme rozlišit dva typy:

Přetrvávající hry
: Obecně se dá říci, že tyto hry nemají žádný přesně určený konec. Zvláště u her pro více hráčů pak herní svět (a případně i samotná hra) existuje a vyvíjí se nezávisle na tom, jestli se jí hráč zrovna účastní nebo ne. Zástupcem této kategorie je například hra *Ingress* (popsána v sekci 2.2.2) a také téměř jakákoliv *sociální – check-in* hra (viz 2.1.2).

Hry s omezeným trváním
: Tyto hry mají pevně daný konec, určený délkou hry v čase nebo ukončovací podmínkou, při jejímž naplnění hráč vítězí či prohrává. Může se také jednat o soutěžení více lidí s cílem dokončit úkol v co nejrychlejším čase.

2.1.2 Rozdělení dle herního principu

Další možné dělení je dle jejich herního principu takto:

Sociální – check-in hry
: Tento typ aplikací představuje spíše sociální sítě s herními prvky, než klasické hry.

Jejich základem jsou herní místa, které lidé navštěvují a následně se z nich nahlašují na server. Za svou aktivitu mohou získávat různé ocenění nebo body, kterými se pak porovnávají s ostatními. Příkladem může být aplikace *Foursquare*, která je popsaná v sekci 2.2.1.

Týmové hry

V těchto typech her proti sobě soutěží herní týmy (týmem může být i jednotlivec) o území, přežití svých herních postav či splnění herního cíle dříve než ostatní. Do této skupiny patří již zmiňovaný *Ingress*, různé *šifrovací hry* a podobně.

Hry pro jednoho hráče

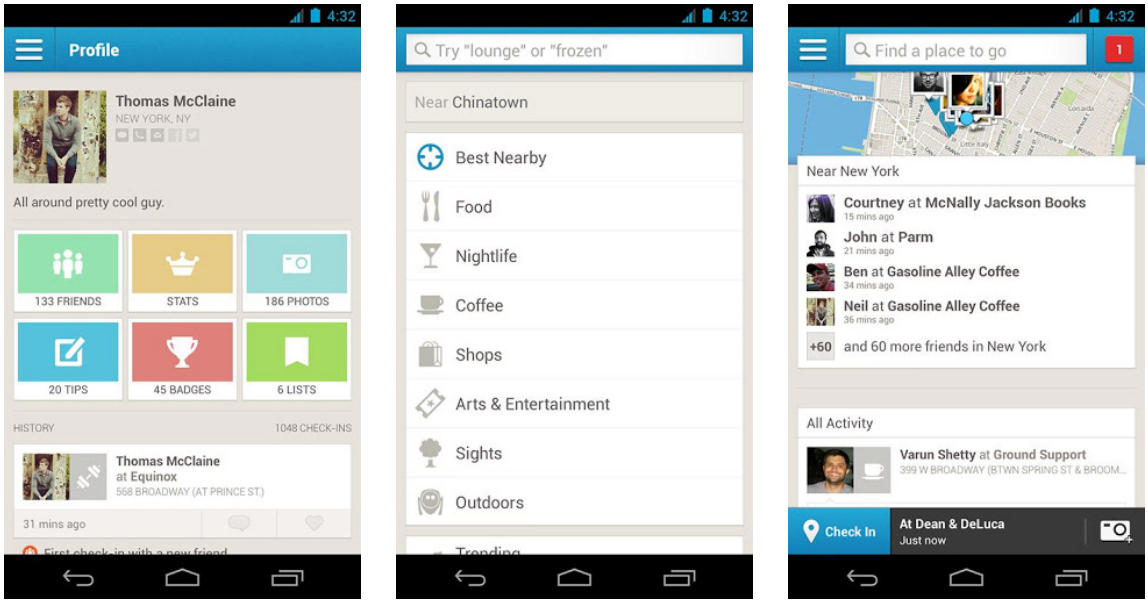
Do této skupiny patří všechny hry, které může hrát jediný člověk, kterého doplňuje virtuální herní svět s případnými počítačem řízenými protivníky. Také se může jednat o hry jako *Geocaching* [17] (kde lidé hledají ukryté předměty) či jiné orientační hry.

2.2 Příklady geolokačních služeb a her pro Android

Zde si uvedeme několik oblíbených aplikací, jenž závisí na poloze uživatele, a popíšeme si jak fungují.

2.2.1 Foursquare

Foursquare [3] je sociální sít s více než 30 miliony uživatelů celosvětově. Její přidanou hodnotou je zaznamenávání navštívených podniků, restaurací, obchodů, kin, dopravních zastávek a dalších veřejných míst. Foursquare je rozšířený po celém světě a i jeho databáze je velmi obsáhlá - navíc nové místa může přidávat každý.



Obrázek 2.1: Náhledy obrazovek aplikace Foursquare.[1]

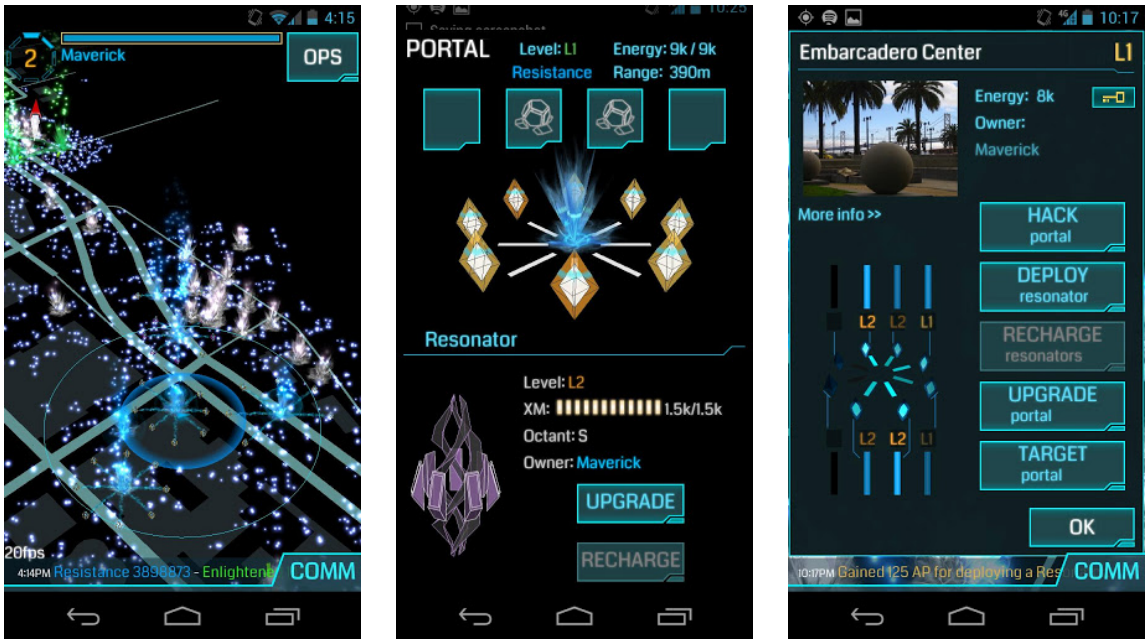
[1]Převzato z https://play.google.com/store/apps/details?id=com.joelapenna.foursquared

Vždy, když člověk přijde na nějaké místo, má zde možnost nahlásit se (provést *check-in*), a tím získat určitý počet bodů. Za nasbírané body nebo časté navštěvování míst určitého typu získává ocenění, kterými se pak porovnává s ostatními lidmi. Uživatel, který se na konkrétním místě vyskytuje nejčastěji, se stává jeho pomyslným starostou (anglicky *mayor*).

Místa se dají komentovat, hodnotit, přidávat k nim fotografie. Lidé mohou zjistit kam jejich přátelé nejčastěji chodí a objevovat různé zajímavé oblasti.

2.2.2 Ingress

Ingress [13] je relativně nová (vytvořena v roce 2012) online týmová geolokační hra využívající prvky virtuální reality. Hráči jsou v ní rozděleni do dvou frakcí, které zjednodušeně řečeno navzájem bojují o území na Zemi. Po celém světě jsou rozmístěny virtuální *portály*, které jsou přiřazené reálným místům jako jsou veřejné sochy, knihovny, pošty a jiné významné budovy. Hráči se pak snaží tyto portály obsazovat a spojovat je do větších skupin (polí). Cílem je pak obsadit co nejvíce těchto portálů. Součástí hry je pak i komplexní příběh, který je postupně odkrýván na stránkách autora. Kdokoliv může také zaslat žádost o přidání nového portálu.



Obrázek 2.2: Náhledy obrazovek aplikace Ingress.[2]

2.2.3 TrailHit

TrailHit [16] je geolokační hra pro jednoho hráče, ve které se člověk snaží najít virtuální zařízení, které má zachránit svět. Aby jej hráč našel, musí nejdříve získat 16 pergamenů (anglicky *scrolls*), kde se každý z nich skládá z jednotlivých stop (anglicky *trails*). Stopy může hráč najít reálným pohybem po okolí, ty v jeho blízkosti jsou zobrazeny na herní mapě. Také může využít režim, kde se po mapě pohybuje pouze šipkami, nebo může tyto

[2] Převzato z https://play.google.com/store/apps/details?id=com.nianticproject.ingress

stopy nakoupit v herním obchodě. U každého pergamenu je následně nutné vyřešit určitou hádanku.

Obrázek 2.3: Náhledy obrazovek aplikace TrailHit.[3]

[3]Převzato z https://play.google.com/store/apps/details?id=com.f5tbl.trailhit

Kapitola 3

Platforma Android

Android je aktuálně nejrozšířenější mobilní operační systém. Je vyvíjený firmou Google, založený na linuxovém jádru, má otevřený zdrojový kód a aplikace pro něj jsou psané výhradně v jazyce Java. Ve speciálních případech je možné využít i psaní nativního kódu v C/C++, např. pro výpočetně náročné algoritmy, fyzikální simulace, zpracování signálů a podobně. Více o tomto je k nalezení v oficiální příručce[1].

K vývoji jsou dostupné vývojářské nástroje (anglicky *Android developer tools*), jejichž hlavní součástí je integrované vývojové prostředí Eclipse, uzpůsobené k vývoji pro Android. Podporuje psaní v jazyce Java i C/C++, rozšířenou práci s XML soubory (zdroji aplikace), umožňuje snadný návrh grafického rozhraní s možnostmi jeho zobrazení pro různé verze a velikosti displejů. Dále jsou zde nástroje pro ladění aplikace na fyzickém zařízení nebo na emulátoru s vytvořeným virtuálním zařízením. [7]

3.1 Základní stavební prvky aplikace

Mezi základní prvky aplikace patří soubor *Android manifest*, *resources*, samotný *zdrojový kód* programu a případně další použité *knihovny*. Ve zdrojovém kódu je pak samotná aplikace (třída *Application*), jednotlivé obrazovky (třída *Activity*), služby běžící v pozadí (třída *Service*) a další. V této sekci si je podrobněji popíšeme.

3.1.1 Android Manifest

Soubor *AndroidManifest.xml* [5] se nachází v kořenové složce projektu a obsahuje všechny důležité informace, které systém potřebuje znát ke spuštění aplikace. Musí zde být uvedeny všechny komponenty aplikace (*activity, services, content providers* a *broadcast receivers*), požadavky o přístup ke konkrétním funkcím systému (např. přístup k internetu, zjištění aktuální polohy…), informace o vyžadovaných funkcích zařízení (např. existence kamery, velikost displeje…), minimální podporovaná verze Android API a další.

3.1.2 Resources

Kromě samotného kódu obsahuje aplikace i *zdroje* (anglicky *resources*) [6], mezi které patří obrázky, zvukové soubory, XML soubory s definicemi layoutů, stylů, menu, barev, a dalších. Zdroje jsou uloženy v adresáři *res* a jeho podsložkách. Pro každý zdroj je vytvořen specifický identifikátor, kterým se na něj dá odkazovat v kódu aplikace nebo v dalších XML souborech.

[1] K nalezení na http://developer.android.com/tools/sdk/ndk/index.html

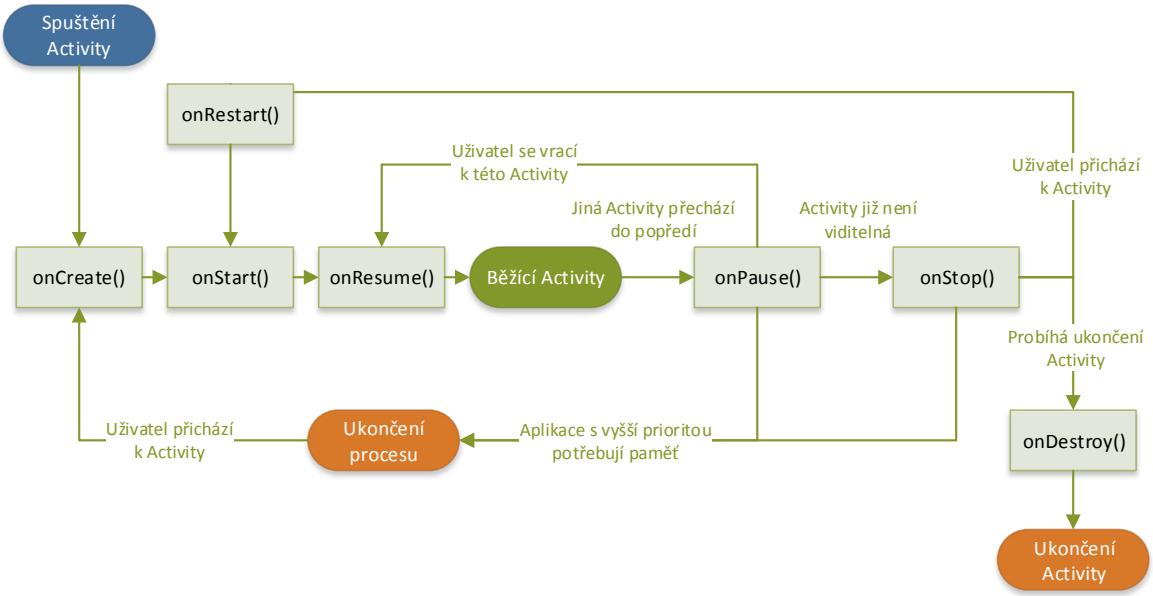
Poskytnutím alternativních zdrojů je možné optimalizovat aplikaci pro různé zařízení a jejich konfigurace (například texty pro různé jazykové mutace či rozdílné layouty pro různé velikosti displeje).

3.1.3 Activity

Aktivita [4] reprezentuje jednu obrazovku uživatelského rozhraní. Může se jednat například o seznam kontaktů nebo zobrazení mapy. Obrazovka je zpravidla zobrazena přes celý displej, ale může to být i menší, plovoucí dialog.

Celá aplikace sestává z několika aktivit, z nichž jedna je zvolena jako hlavní – ta je zobrazena jako výchozí při běžném spuštění aplikace. Jednotlivé aktivity jsou na sobě nezávislé, kterákoli z nich může spustit jinou aktivitu a to ať už je součástí stejné nebo i jiné aplikace (pokud je to povoleno v *manifestu* dané aplikace).

Životní cyklus aktivity je vyznačen na obrázku 3.1



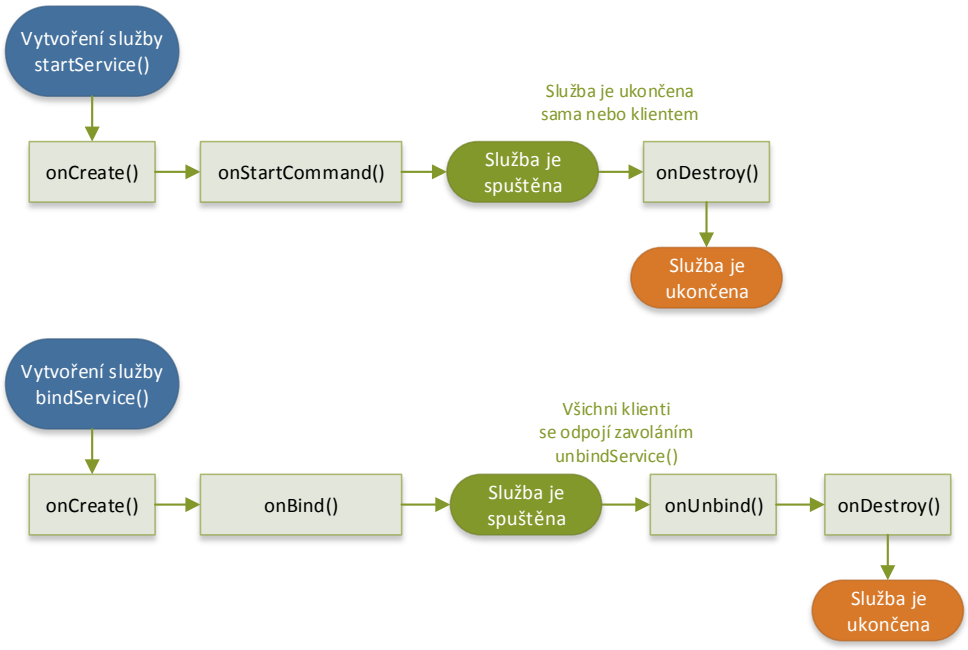
Obrázek 3.1: Životní cyklus aktivity

3.1.4 Service

Služba [11] je prvek aplikace, který běží na pozadí a zpravidla vykonává nějakou dlouhotrvající operaci. Může se jednat o stahování dat, přehrávání hudby a podobně. Služby nemají vlastní uživatelské rozhraní.

Služba se spouští buď voláním *startService()* nebo *bindService()* (pro umožnění komunikace s klienty). Po jejím spuštění běží v pozadí až dokud není ukončena voláním *stopSelf()*, *stopService()* nebo dokud se neodpojí všichni klienti voláním *unbindService()* (případně dokud není potřeba paměť pro prioritnější úlohy).

Životní cyklus služby je vyznačen na obrázku 3.2



Obrázek 3.2: Životní cyklus služby

3.1.5 Knihovny

Pro oddělení znovupoužitelných částí kódu je možné využít tzv. *Library Projects* [8], tedy knihoven. Tento typ projektu nemůže být přímo nainstalován na koncové zařízení, ale místo toho je vložen do *.apk* souboru s aplikací při její kompilaci. Struktura projektu je prakticky shodná s klasickou aplikací, je tu však pár omezení, týkajících se jeho použití.

Všechny použité knihovny je nutné uvést a přidat do projektu aplikace a to buď ve formě zkompilovaného *.jar* souboru nebo jako projekt se zdrojovými kódy knihovny. Aplikace může používat více knihoven a i ty mohou používat další knihovny. Ve výsledku existuje pouze jedna složka např. se *zdroji*, takže při existenci duplicit bude použit pouze zdroj z aplikace a v případě duplicit mezi knihovnami ten, jehož knihovna má nastavenu vyšší prioritu. *Asset resources* (zdroje ve složce *assets*) nejsou v knihovnách povoleny a musí být umístěny až v samotné aplikaci.

3.2 Podpora 3D grafiky

Android podporuje *Open Graphics Library* [9] (OpenGL), což je multiplatformní grafické API, které specifikuje základní programové rozhraní pro práci s 3D grafikou. Konkrétní podporované verze jsou OpenGL ES[2] 1.0, 1.1 a od verze Androidu 2.2 je podporováno i OpenGL ES 2.0.

Pro využití OpenGL ES API je potřeba dvou základních tříd:

GLSurfaceView

Tato třída představuje komponentu, která umožňuje kreslení a manipulaci s objekty

[2]OpenGL ES - Specifikace OpenGL zaměřená na vestavěné systémy.

prostřednictvím OpenGL API metod. Po vytvoření její instance je nutné nastavit jí vlastní *Renderer* metodou *GLSurfaceView.setRenderer()*. Pro použití OpenGL ES verze 2.0 je nutné následně zavolat metodu *GLSurfaceView.setEGLContextClientVersion(2)*.

GLSurfaceView.Renderer
Toto rozhraní definuje základní metody nutné pro kreslení nad *GLSurfaceView*. Konkrétně se jedná o:

- **onSurfaceCreated()**: Tuto metodu systém volá pouze při vytváření GLSurfaceView komponenty. Hodí se pro prvotní nastavení prostředí OpenGL nebo k inicializaci grafických objektů.
- **onDrawFrame()**: Systém volá tuto metodu při každém překreslení komponenty, sem tedy patří hlavní vykreslovací kód.
- **onSurfaceChanged()**: Tato metoda je zavolána pokaždé, když nastane změna geometrie. Např. při změně velikosti komponenty nebo orientace zařízení.

Vzhledem k tomu, že OpenGL předpokládá klasický čtvercový souřadný systém, zatímco velikosti displejů se značně liší, je potřeba toto řešit vhodným nastavením projekce a kamerového zobrazení. Více o tomto je k nalezení v příručce [9].

Ne všechny zařízení navíc podporují verzi API 2.0 a všechny možnosti komprese textur, takže je nutné tyto požadavky uvést v *Manifestu* aplikace.

3.2.1 Vykreslování

Nejprve je nutné vytvořit si požadované objekty k vykreslení. Objekty jsou tvořeny jednotlivými tvary, jejichž vrcholy mají definované souřadnice v 3D prostoru. Vrcholy těchto bodů jsou definovány v pořadí proti směru hodinových ručiček, což určuje jejich přední stranu. Je možné vypnout vykreslování zadních stran tvarů, což s sebou přinese značné urychlení vykreslování. Pro další zefektivnění je také možné aktivovat vykreslování pouze na vyžádání.

Pro samotné vykreslení s použitím OpenGL ES 2.0 je nejdříve potřeba definovat:

- **Vertex Shader** pro vykreslování vrcholů tvaru,
- **Fragment Shader** pro vykreslování plochy tvaru barvou nebo texturou a
- **Program**, což je objekt, který obsahuje výše zmíněné shadery, které se následně použijí pro vykreslení tvaru (tvarů).

Shadery obsahují kód *OpenGL Shading Language* (GLSL), který musí být před použitím zkompilován. Shadery jsou vloženy do *Programu*, který je následně předán hlavnímu prostředí a zbývá jen určit co a jakým způsobem se má vykreslit.

3.3 Senzory

Senzory [10] můžeme chápat jako funkce, které podávají informace o stavu zařízení nebo jeho okolí. Většina zařízení se systémem Android obsahuje senzory pro měření pohybu, polohy či různých vlastností okolního prostředí. Data ze senzorů mohou být využita k různým účelům jako je navigace uživatele, ovládání her, podpora rozšířené reality a podobně. S vývojem

nových technologií a hardwaru se navíc zvyšuje jejich přesnost a objevují se i nové typy senzorů.

Třída *Sensor* představuje dva typy senzorů: *surové* (či *hardwarové*) a *syntetické* (či *softwarové*). První typ předává surové data přímo z konkrétní hardwarové komponenty v zařízení. Druhý typ představuje abstraktní vrstvu mezi kódem aplikace a komponentami zařízení. Buď se jedná o kombinaci dat z několika surových senzorů, nebo o předzpracování dat senzoru pro jejich jednodušší využití. Pokud je to možné, je doporučeno využívat radši syntetické senzory. [15]

Jaké konkrétní typy senzorů můžeme využít a jakým způsobem získat jejich data je popsáno dále v této sekci.

3.3.1 Zjištění polohy uživatele

Android poskytuje možnost zjištění aktuální polohy uživatele pomocí různých *služeb*. Ty se od sebe liší přesností, rychlostí, spotřebou baterie, apod. Dostupné jsou tyto možnosti, které je možné kombinovat:

GPS modul
: GPS modul nabízí nejpřesnější výsledky (přesnost až kolem 5 metrů), ale rychle spotřebovává baterii a také trvá delší dobu (několik sekund až minut) než dojde ke zjištění polohy. Další nevýhodou je, že jej nelze využít ve vnitřních prostorách nebo pří špatné viditelnosti oblohy (např. pokud jsou v okolí vysoké budovy).

Mobilní vysílače a WiFi sítě
: Tato možnost využívá vysílačů mobilní sítě a WiFi v okolí. Funguje jak uvnitř, tak venku, reaguje rychleji a spotřebovává méně baterie. Na druhou stranu její přesnost je nižší (stovky metrů a více).

Pasivní zjištění
: Tento způsob sám aktivně polohu nevyhledává, ale čeká na zjištění polohy jinou aplikací. Přesnost závisí na použité službě dané aplikace, spotřeba baterie navíc není prakticky žádná, ale může trvat velmi dlouhou dobu, než si informace o aktuální poloze jiná aplikace vyžádá.

O zjišťování a správu polohy uživatele se stará třída *LocationManager*. Poskytuje tři základní funkce:

- Zjistit poslední dostupnou informaci o poloze uživatele (metoda *getLastKnownLocation()*).
- Zaregistrovat či zrušit příjem průběžných aktualizací polohy (metoda *requestLocationUpdates()* resp. *removeUpdates()*).
- Zaregistrovat či zrušit zasílání *Intentu*, když se uživatel dostane do blízkosti určitého místa, určeného souřadnicemi a poloměrem v metrech (metoda *addProximityAlert()* resp. *removeProximityAlert()*).

Pro příjem aktualizací polohy je nutné vytvořit třídu implementující rozhraní *LocationListener* a zaregistrovat ji výše uvedeným způsobem. Nejdůležitější je pak metoda *onLocationChanged()*, která jako parametr přijímá objekt *Location*, který obsahuje všechny důležité informace (souřadnice, přesnost, informace o použité službě, apod.). Toto rozhraní obsahuje ještě další tři metody informující o změně stavu služby.

3.3.2 Senzory pohybu

Tyto senzory měří akceleraci a rotaci ve třech osách. Jsou vhodné pro měření pohybu zařízení jako je naklánění a otáčení. Tento pohyb je většinou způsoben přímou interakcí uživatele, ale může se jednat také o vliv prostředí, ve kterém se uživatel právě nachází (např. dopravní prostředek).

Konkrétní senzory jsou uvedeny v tabulce 3.1. Většina zařízení obsahuje alespoň akcelerometr, mnoho jich má také gyroskop.

3.3.3 Senzory polohy

Polohové senzory určují pozici zařízení v rámci reálného světa nebo přiblížení vůči jinému objektu (například obličej uživatele při telefonování).

Konkrétní senzory jsou uvedeny v tabulce 3.2. Zařízení většinou obsahují senzor geomagnetického pole a přiblížení.

3.3.4 Senzory prostředí

Tyto senzory umožňují sledovat různé vlastnosti okolního prostředí. Narozdíl od pohybových a polohových senzorů, jejichž data často vyžadují filtrování, tyto senzory ani filtrování ani jiné zpracování zpravidla nepotřebují.

Konkrétní senzory jsou uvedeny v tabulce 3.3. Tyto senzory (s výjimkou světelného, který se používá pro automatickou změnu jasu displeje) na zařízeních většinou nebývají dostupné.

3.3.5 Zpracování senzorových dat

Pro přístup k senzorům Android poskytuje několik tříd a rozhraní. Hlavní z nich je třída *SensorManager*, která poskytuje základní metody pro získání informací o jednotlivých senzorech (např. jejich dostupnost, rozsah, využití baterie) a zaregistrování či zrušení monitorování změn jejich hodnot (metodou *registerListener()* resp. *unregisterListener()*). Dostupnost jednotlivých senzorů závisí jak na fyzické dostupnosti na hardwaru, tak i na použité verzi systému.

Pro příjem dat ze senzorů je nutné vytvořit třídu, která implementuje rozhraní *SensorEventListener* a poté ji zaregistrovat výše uvedeným způsobem. Samotný příjem dat pak probíhá v metodě *onSensorChanged()*, která jako parametr získává objekt typu *SensorEvent* se všemi informacemi. Ty se liší v závislosti na konkrétním senzoru. Téměř všechny pohybové a polohové senzory vrací pole hodnot pro všechny osy, zatímco senzory rozhraní pouze jednu konkrétní hodnotu. Metodou *onAccuracyChanged()* je pak možné sledovat změnu přesnosti měření.

3.4 Tvorba aplikací s prvky rozšířené reality

Pro podporu rozšířené reality v aplikacích existuje mnoho knihoven[3], které je možné využít. Liší se jednotlivými funkcemi dle typu aplikace pro kterou jsou určeny a také svou licencí (či cenou). V této sekci si představíme dvě oblíbené knihovny, které jsou dostupné zdarma pro nekomerční použití.

[3]Krátký přehled knihoven s podporou rozšířené reality lze nalézt na http://en.wikipedia.org/wiki/List_of_augmented_reality_software

Senzor	Typ	Popis	Jednotka
TYPE_ACCELEROMETER	HW	Měří velikost zrychlení zařízení ve všech třech osách, včetně gravitační síly.	m/s^2
TYPE_GRAVITY	SW/HW	Měří gravitační sílu, která působí na zařízení ve všech třech osách.	m/s^2
TYPE_GYROSCOPE	HW	Měří rychlost rotace zařízení kolem všech tří os.	rad/s
TYPE_LINEAR_ACCELERATION	SW/HW	Měří velikost zrychlení zařízení ve všech třech osách, bez gravitační síly.	m/s^2
TYPE_ROTATION_VECTOR	SW/HW	Zjišťuje orientaci zařízení pomocí vektoru rotace.	žádná

Tabulka 3.1: Přehled dostupných senzorů pohybu

Senzor	Typ	Popis	Jednotka
TYPE_MAGNETIC_FIELD	HW	Měří okolní geomagnetické pole pro všechny tři osy.	μT
TYPE_ORIENTATION	SW	Měří stupně rotace zařízení ve všech třech osách.	stupně
TYPE_PROXIMITY	HW	Měří přiblížení obrazovky zařízení k jinému objektu.	cm

Tabulka 3.2: Přehled dostupných senzorů polohy

Senzor	Typ	Popis	Jednotka
TYPE_AMBIENT_TEMPERATURE	HW	Měří teplotu okolního prostředí.	μT
TYPE_LIGHT	HW	Měří hladinu okolního světla.	lx
TYPE_PRESSURE	HW	Měří okolní tlak vzduchu.	hPa nebo mb
TYPE_RELATIVE_HUMIDITY	HW	Měří relativní vlhkost prostředí.	%
TYPE_TEMPERATURE	HW	Měří teplotu zařízení. Liší se v závislosti na konkrétním zařízení.	°C

Tabulka 3.3: Přehled dostupných senzorů prostředí

Použití konkrétní knihovny je jednoduché, většinou stačí připojit její zdrojové kódy (nebo soubor *.jar*, který je obsahuje již zkompilované) do svého projektu a v tu chvíli už je možné využít všech jejich funkcí.

3.4.1 ARToolkit

ARToolkit [1] je multiplatformní framework pro tvorbu aplikací s rozšířenou realitou. Vytvořil jej *Hirokazu Kato* v roce 1999 a dále je vyvíjen jako Open Source s možností prodeje komerčních licencí společností *ARToolWorks*. Je napsaný v jazyce *C* a podporuje systémy *Windows*, *Linux*, *Mac OS X* a další. Uživateli byl přepsán do mnoha dalších jazyků a operačních systémů. Například pro jazyk Java a platformu Android existuje projekt *AndAR*[4].

Tento framework umožňuje použití rozšířené reality založené na *značkách* (*markerech*) umístěných v reálném světě. Má rychlou a nenáročnou detekci jejich sledování a pro vykreslování scény využívá *OpenGL*.

3.4.2 DroidAR

DroidAR [2] je aktuálně jeden z nejpoužívanějších Open Source frameworků (vydaný pod licencí *GNU GPL v3*) pro tvorbu aplikací s rozšířenou realitou na platformě Android. Byl vytvořen roku 2010 *Simonem Heinenem* a následně vyvíjen komunitou a skupinou *bitstars*.

Framework podporuje zobrazení virtuální scény skrz pohled kamery, který je založený buď na poloze uživatele nebo také použitím *značek* (*markerů*) umístěných v reálném světě. Tyto dva přístupy je možné kombinovat. Framework je možné využívat na zařízeních s verzí Android API 1.6 a vyšší. Pro rozšířenou realitu založenou na poloze využívá GPS, senzory magnetického pole, akcelerometr a gyroskop. Pro zpřesnění údajů framework podporuje také detekci kroků uživatele.

Ukázku herní scény zkombinované s pohledem z kamery demonstruje obrázek 3.3.

Obrázek 3.3: Ukázka aplikace používající framework DroidAR.[5]

[4] AndAR je k nalezení na http://code.google.com/p/andar/
[5] Převzato z http://droidar.blogspot.cz/2012/10/droidar-screenshots.html

Kapitola 4

Návrh frameworku

V kapitole 2 jsme si popsali principy geolokačních her a uvedli i některé konkrétní příklady. Protože tyto hry mohou být značně rozdílné, není možné (a ani nutné), aby framework podporoval všechny způsoby a možnosti, ale je potřeba určit si konkrétní směr, kterým se bude ubírat. Rozhodl jsem se, že framework bude podporovat pouze hry pro jednoho hráče. Pro tento typ her bude univerzálně použitelný a bude umožňovat jednoduchou změny své konfigurace. Z tohoto důvodu by od sebe měla být oddělena definice samotné hry, uživatelské rozhraní a zbývající aplikační logika.

Framework by se měl starat o načítání hry a správu jejího průběhu, zpracovávat informace o aktuální poloze hráče, uchovávat a aktualizovat herní čas, umožňovat zpětnou vazbu (zvukové efekty, vibrace, přepínání activity, apod.), rozesílat a zpracovávat herní události, umožnit interakci s reálným světem – podpora skenování kódů (QR či jiných), sledovat příchody a odchody uživatele z definovaných oblastí a podobně.

Definice hry by měla být oddělena ve zvláštním souboru (souborech). Co všechno by zde mělo být možné uvést je popsáno v sekci 4.1.

4.1 Herní scénář

Popis všech herních objektů a jejich vzájemného chování je uložen v tzv. *herním scénáři*. Herní objekty mohou být různého typu. V mém případě jsem zvolil *herní zprávy*, *oblasti*, *proměnné*, *herní události* neboli *reakce* a také *spouštěče*, které tyto reakce za konkrétních *podmínek* aktivují. Jednoduchý přehled objektů je v tabulce 4.1.

Objekt	Má ID	Popis
Zpráva	ano	Informace, které hráči přicházejí v průběhu hraní, např. uvítací zpráva.
Oblast	ano	Místo v reálném světě, u kterého chceme sledovat, jestli se v něm uživatel právě nachází.
Proměnná	ano	Uchovává a umožňuje pracovat s určitou hodnotou. Např. skóre nebo informace o (ne)splnění herního úkolu.
Reakce	ano	Akce, která se má vykonat. Např. aktivace vibrací či odeslání herní události.
Spouštěč	ne	Na základě výskytu určité herní události (a v případě splnění dalších podmínek) aktivuje konkrétní reakci.

Tabulka 4.1: Přehled herních objektů

Většina herních objektů má vlastní *identifikátor* (ID), který je unikátní v rámci skupiny daného typu objektů. Tímto způsobem se na objekty dá odkazovat a také mezi nimi vyhledávat. Jedinou výjimku tvoří *spouštěče a podmínky*, na které není potřeba se odkazovat.

Jednotlivé části scénáře si nyní blíže popíšeme.

4.1.1 Základní informace

Každý scénář musí obsahovat základní údaje o sobě a svém autorovi. Mezi tyto informace patří název, krátký popis, datum vytvoření, verze, jméno autora a případně některé další údaje. Do budoucna by mohl scénář obsahovat i číslo verze frameworku, pro který je určen.

4.1.2 Zprávy

Herní zprávy představují informace, které hráči přicházejí v průběhu hraní. Může se jednat o uvítací zprávy, které se objeví s příchodem hráče na určité místo nebo to mohou být informace o vývoji herního děje, úkoly nutné ke splnění hry a podobně.

Každá zpráva obsahuje svůj *titulek* a pak samotný *obsah*, který není co se týče formátu nijak omezený – záleží pak na implementaci konkrétní aplikace, jak jej uživateli předá. Framework by však měl podporovat práci alespoň s čistě *textovými soubory*. Interně by zprávy měly obsahovat také unikátní *identifikátor* (kterým se na zprávu bude dát odkazovat), informaci o *čase svého doručení*, svůj *aktuální stav* (nedoručeno, doručeno ale ještě nepřečteno, přečteno, smazáno) a *štítek*, který bude určovat typ zprávy – opět pak záleží na samotné aplikaci jak jej využije (pokud vůbec).

Mělo by být možné vyhledat zprávu dle jejího identifikátoru a také všechny doručené (a nesmazané) zprávy s konkrétním štítkem.

4.1.3 Oblasti

Herní oblasti se dají považovat za základní stavební kámen geolokačních her. Právě příchod uživatele na konkrétní místo (nebo jeho následný odchod) vyvolá určitou událost, na kterou musí hra vhodným způsobem zareagovat (konkrétní reakce jsou popsány v sekci 4.1.6).

Každá oblast má vlastní identifikátor a určený prostor, který zabírá. Tento prostor může být definován jako *kruh* se středem v určitém bodě a svým poloměrem nebo jako *mnohoúhelník* složený z několika hraničních bodů. Jednotlivé body jsou určeny svými zeměpisnými souřadnicemi [19].

4.1.4 Proměnné

Proměnné ze scénáře uchovávají určitou hodnotu a dovolují s ní dále pracovat. Proměnná může vyjadřovat například počet navštívených území, herní skóre uživatele a podobně. Hodnotu proměnné lze za běhu jednoduše měnit a tato změna může následně vyvolat určitou reakci (viz 4.1.6).

Jednotlivé proměnné jsou identifikované svým názvem, typem (jenž určuje, jakých hodnot může proměnná nabývat) a případným dalším omezením její minimální a maximální hodnoty. Také je určena její výchozí hodnota, kterou proměnná nabývá při spuštění hry.

4.1.5 Spouštěče a podmínky

Spouštěč je aktivní prvek scénáře, který je navázán na jednu z herních událostí a při jejím výskytu aktivuje zvolenou reakci (viz sekce 4.1.6). Aktivace reakce může být navíc omezena

dalšími parametry, které spouštěč obsahuje. Jedním z nich je nastavení omezení počtu jeho spuštění a druhým jsou pak samostatné *podmínky*, které kontrolují hodnotu herních *proměnných*. Každá podmínka je složena z operátoru porovnání, identifikátoru proměnné a porovnávané hodnoty.

Z hlediska podmínek můžeme rozlišovat tři druhy spouštěčů:

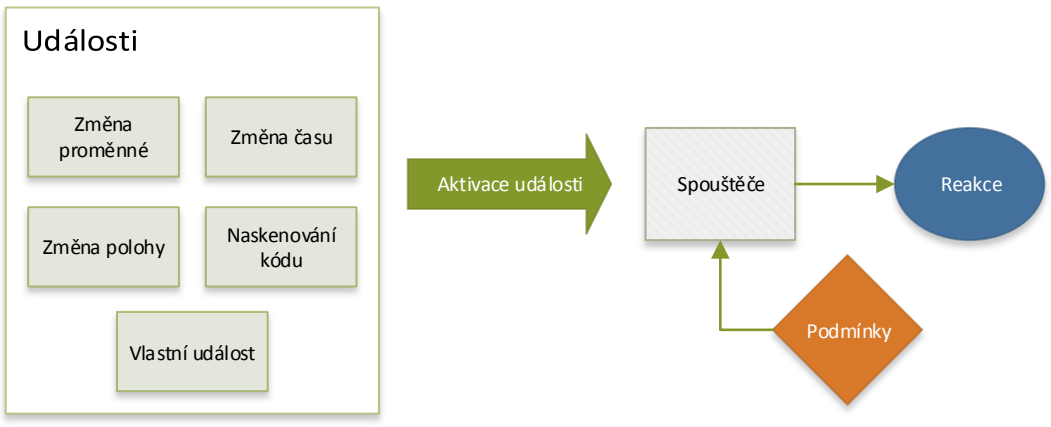
- **bez podmínek**,
- s více podmínkami, kde **alespoň jedna musí platit**,
- s více podmínkami, kde **všechny musí platit**.

Jak už bylo řečeno, spouštěče čekají na výskyt nějaké herní události. Těmito událostmi mohou být:

- **příchod** (nebo odchod) uživatele **do oblasti**,
- **změna hodnoty proměnné**,
- **změna času** - herní čas je speciální proměnná uvnitř frameworku, která se automaticky aktualizuje během hry,
- **vlastní událost** - vyvolána reakcí nebo samotnou aplikací,
- **naskenování kódu** - pomocí fotoaparátu zařízení uživatel naskenuje čárový kód, což vyvolá tento spouštěč.

4.1.6 Reakce

Reakce už svým názvem značí, že vykonávají nějakou akci jako odpověď na určitý impulz. Impulzem bývá v tomto případě spouštěč, který je popsán v sekci 4.1.5. Tento princip demonstruje obrázek 4.1.



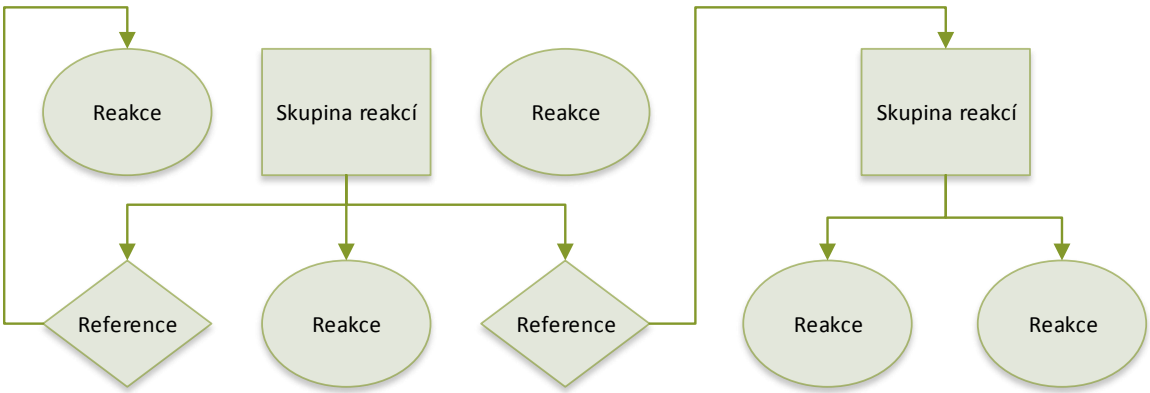
Obrázek 4.1: Souvislost herních událostí, spouštěčů a reakcí.

Každá reakce obsahuje vlastní identifikátor (až na výjimky, viz dále) a případné další hodnoty, které závisí na svém typu. Rozlišujeme několik základních typů reakcí:

- **zvuková reakce** - přehraje zvolený audio záznam,
- **vibrační reakce** - aktivuje vibrace na pevně danou dobu,
- **herní zpráva** - tato reakce způsobí doručení (zviditelnění) konkrétní herní zprávy (určenou jejím identifikátorem),
- **herní událost** - vyvolá jednu z předdefinovaných událostí nebo událost vlastní,
- **aktivita** - způsobí spuštění konkrétní aktivity (určenou jejím celým názvem),
- **změna proměnné** - aplikuje matematickou operaci se zvoleným parametrem pro změnu hodnoty proměnné (určenou jejím identifikátorem).

Aby nedocházelo k redundanci kódu spouštěčů, je nutné zavést také dvě speciální reakce, které zároveň zpřehlední jejich složitější kombinace, jak ukazuje obrázek 4.2:

- **skupina reakcí** - tento typ představuje složku seskupující několik souvisejících reakcí, které jsou vyvolávány najednou. Například můžeme chtít přehrát zvuk, aktivovat vibrace a změnit hodnotu proměnné v rámci jedné akce. Samotné seskupené reakce nemají vlastní identifikátory, ten má pouze jejich skupina.
- **reference na reakci** - abychom zabránili případným duplicitám samotných reakcí (např. jeden zvuk se může přehrát v kombinaci s různými reakcemi), je vhodné mít možnost odkázat se na již definovanou reakci. Aby tyto reference dávaly smysl, mohou být definovány pouze uvnitř skupiny reakcí a musí odkazovat na samostatnou reakci nebo jinou skupinu. Odkazovat se na jednotlivé reakce uvnitř skupin nelze už jen z toho důvodu, že tyto reakce nemají vlastní identifikátor. U tohoto typu reakce bude nutné kontrolovat případné kruhové závislosti nebo odkazování na sebe sama.



Obrázek 4.2: Ukázka speciálních reakcí - skupiny a reference.

4.2 Uživatelské rozhraní

Framework neobsahuje vlastní uživatelské rozhraní, ale pouze aplikační logiku. To dává autorovi výsledné aplikace možnost vytvoření uživatelského rozhraní přesně dle potřeb konkrétní hry, aniž by byl něčím omezován.

Kapitola 5

Implementace a testování

Tato kapitola se věnuje implementaci a následnému testování frameworku. Je zde popsána jeho struktura, některé důležité třídy a funkce, způsob uložení a načítání scénáře a podobně. V sekci 5.7 je uvedeno co vše je potřeba k použití tohoto frameworku v jiné aplikaci. Sekce 5.8 pojednává mimo jiné o vytvořené testovací aplikaci.

5.1 Struktura frameworku

Zdrojové kódy frameworku jsou rozmístěny v několika základních balíčcích: *constants*, *game*, *scenario* a *utils*.

Constants
: V tomto balíčku je pouze jedna třída *Constants*, která obsahuje některé, frameworkem používané, konstanty.

Game
: Zde je uložena herní služba, třídy reprezentující herní stavy a události, rozhraní pro příjem a distribuci těchto událostí a třída starající se o samotnou distribuci.

Scenario
: Tento balíček obsahuje několik abstraktních tříd pro použití jako předky dalších herních objektů, třídu se samotným scénářem a informacemi o něm. Dále jsou zde balíčky pro jednotlivé herní objekty: *area*, *hook*, *message*, *reaction* a *variable*. Balíček *parser* obsahuje třídu pro načítání scénáře z XML souboru a v balíčku *helpers* jsou některé pomocné objekty.

Utils
: Zde jsou uloženy pomocné třídy s obecně užitečnými funkcemi. Například třída *GPoint*, která reprezentuje geografické souřadnice a usnadňuje výpočet vzdálenosti mezi dvěma body v metrech.

5.2 Herní scénář

Herní scénář je reprezentován třídou *Scenario*, která obsahuje kromě základních informací (třída *ScenarioInfo*) také všechny hlavní herní objekty. Jednotlivé mapy *oblastí*, *proměnných*, *reakcí* a *zpráv*, seznam *spouštěčů*, instance objektu *SoundPool* pro načítání a přehrávání zvuků, speciální objekty *TimeHookable*, *ScannerHookable* a *EventHooktable*, které

se starají o distribuci daných událostí připojeným spouštěčům, a také aplikační kontext a *GameHandler* pro zasílání herních událostí.

Herní objekty se dají vyhledávat dle jejich identifikátoru. Metoda *getVisibleMessages()* vrací seznam všech zpráv, které jsou pro hráče viditelné (tedy přijaté a nesmazané). Po zavolání některé ze čtveřice metod (o což se stará herní služba) *onTimeUpdate()*, *onLocationUpdate()*, *onCustomEvent()* a *onScanned()* se aktivují všechny spouštěče připojené k daným objektům.

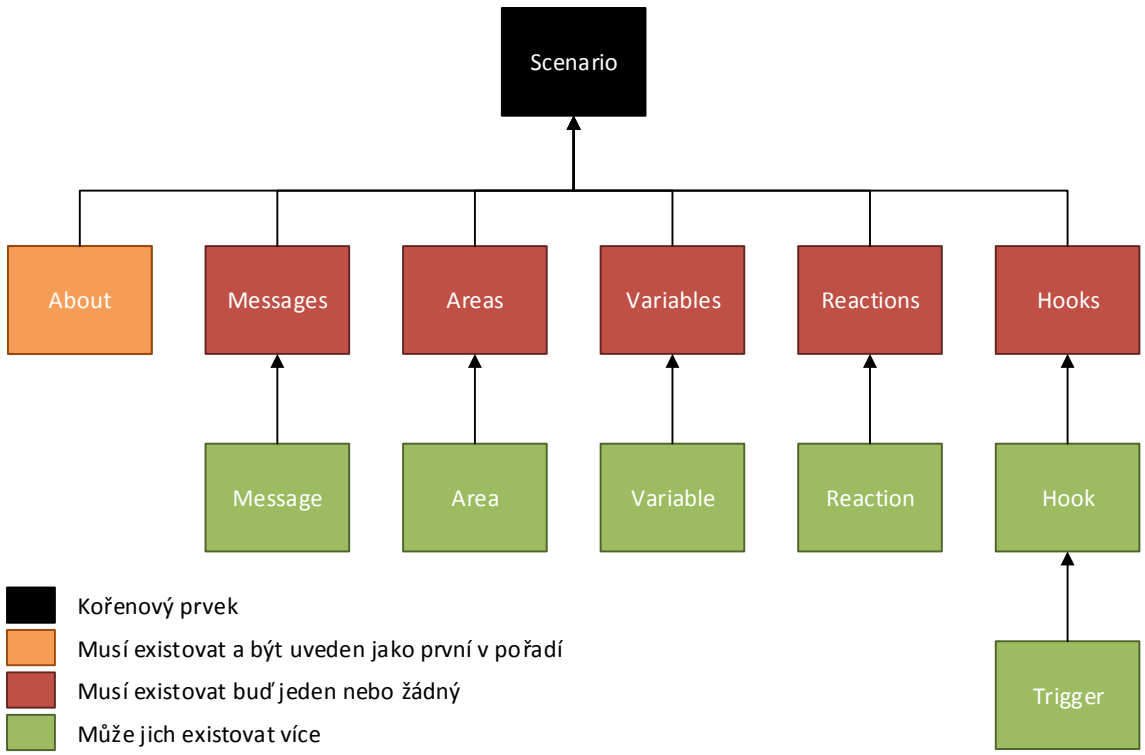
5.3 XML parser scénáře

Pro uložení definice hry jsem zvolil soubor ve formátu XML [14], který je pro člověka jednoduše čitelný a i podpora pro jeho načítání je v jazyce Java standardně k dispozici.

Pro načtení scénáře slouží třída *XmlScenarioParser*, která obsahuje statické metody *fromFile()* (pro načtení souboru z karty zařízení) a *fromAsset()* (pro načtení souboru ze složky *assets* aplikace), které krom *kontextu* přijímají cestu k souboru. Následně vrátí buď novou instanci scénáře se všemi načtenými herními objekty, nebo *null* v případě chyby.

Při načítání je také možno zvolit, aby se načetly jenom základní informace o scénáři (bez herních objektů), což může být využito například pro zobrazení výpisu všech dostupných scénářů dané aplikace.

5.3.1 Struktura scénáře



Obrázek 5.1: Struktura scénáře uloženého v XML souboru.

Obrázek 5.1 naznačuje základní strukturu scénáře. Element <*scenario*> obaluje všechny skupiny objektů (zprávy - *Messages*, oblasti - *Areas*, proměnné - *Variables*, reakce - *Reactions*, spouštěče - *Hooks*) a samotné informace o scénáři (element <*about*>), které musí být navíc uvedeny jako první. Pořadí ostatních skupin je volitelné. V každé skupině jsou pak definovány objekty daného typu.

Konkrétní ukázka scénáře i s přehledem všech podporovaných elementů a jejich atributů je k nalezení v testovací aplikaci a v příloze B.

5.3.2 Průběh načítání

Parsování XML souboru probíhá s využitím *XmlPullParser*u [12], což je vzhledem k jeho nízké paměťové náročnosti (načítá soubor postupně a ne celý najednou jako například *DOM parser*) vhodný způsob pro použití na mobilní platformě.

Samotné načítání pak probíhá v metodě *parse()*, kde se v cyklu prochází celý soubor a kontrolují se jednotlivé elementy. Ty neznámé nebo neočekávané jsou ignorovány a přeskočeny, ale také je o nich zaznamenána informace do konzole. Po načtení základních informací o scénáři je vytvořena instance objektu *Scenario*, kterou metoda na konci vrací. Herní objekty se hned po vytvoření přidávají do tohoto scénáře, který navíc kontroluje, zda-li již objekt s daným identifikátorem neexistuje.

Po načtení celého scénáře se zavolá metoda *onLoaded()* scénáře, která se postará o propojení jednotlivých objektů, spouštěčů a reakcí k sobě. Objekty mají také možnost načíst si potřebné zdroje (např. zvukové soubory) a podobně.

5.3.3 Použití jiného formátu scénáře

Pokud by bylo potřeba načítat scénář z jiného typu souboru, stačí vytvořit novou třídu s novým *parserem* pro daný typ souboru a následně přepsat metodu *loadScenario()* herní služby. Tato metoda se stará o načtení scénáře zvoleným způsobem a následné vrácení jeho instance.

5.4 Herní události

Herní událost představuje třída *GameEvent*. Obsahuje typ události (třída *EventType*) a její hodnotu (jakýkoliv objekt). Framework rozeznává 10 typů herních událostí:

GAME_START
: Spuštění hry (buď právě vytvořené, nebo pozastavené). Odeslání této události aktivuje herní časovač a případně i vyžádání aktualizací polohy.

GAME_PAUSE
: Pozastavení hry. Odeslání této události pozastaví jak herní časovač, tak aktualizace polohy.

GAME_WIN
: Tato událost značí, že uživatel vyhrál a hra končí.

GAME_LOSE
: Tato událost značí, že uživatel prohrál a hra končí.

GAME_QUIT
Zavoláním této události se ukončí celá herní služba.

UPDATED_LOCATION
Při každé aktualizaci polohy je přijata tato událost, jejíž hodnota obsahuje objekt typu *Location* s aktuálními souřadnicemi hráče.

Tato informace je herní službou distribuována scénáři, který projde všechny oblasti a zavolá jejich metodu *updateLocation()*.

UPDATED_TIME
Časovač kromě zaznamenávání a aktualizace herního času také každou sekundu zasílá tuto událost. Její hodnota obsahuje objekt typu *Long* obsahující aktuální délku hry v milisekundách.

Tato informace je také distribuována scénáři, pro případnou aktivaci spouštěčů.

UPDATED_MESSAGES
Událost tohoto typu značí zpřístupnění (označení jako přijaté) herní zprávy.

SCANNED_CODE
Tato událost je přijata vždy, když uživatel úspěšně provede naskenování kódu (viz sekce 5.6). Její hodnota obsahuje právě naskenovaný kód.

Tato informace je také distribuována scénáři, pro případnou aktivaci spouštěčů.

CUSTOM
Tento typ obsahuje hodnotu definovanou ve scénáři nebo aplikaci a její použití záleží na konkrétní implementaci aplikace.

Tato informace je také distribuována scénáři, pro případnou aktivaci spouštěčů.

5.5 Herní služba

Herní služba, reprezentována třídou *GameService*, tvoří jádro celého frameworku. Během jejího spuštění se načte herní scénář, zobrazí systémové oznámení, vyžádá se příjem aktualizací polohy a následně se čeká na startovní událost. Výskyt této události záleží na konkrétní implementaci, může se jednat například o příchod uživatele na konkrétní místo. O ukončení hry a služby se také stará autor aplikace.

Tato služba je používána jako *singleton*[1], požadavky o opakované vytvoření služby (pokud již běží) jsou ignorovány. Vždy je však zavolána metoda *onGameStart()* (viz sekce 5.7) lišící se parametrem, což dovolí reagovat i na tyto situace. Kontrola existence služby se provádí voláním *GameService.isRunning()*. Pokud metoda vrátí *true*, je možné získat instanci služby zavoláním *GameService.getInstance()*.

Prostřednictvím její instance je možné získat přístup k hernímu scénáři, poloze hráče, aktuálnímu a počátečnímu času hry, stavu hry a podobně. Také je možné zaregistrovat příjem herních událostí do vlastní třídy, která implementuje rozhraní *GameEventListener*, zavoláním metody *RegisterListener()*.

[1] Singleton - třída s jedinou instanci a globálním přístupovým bodem

5.5.1 Stavy hry

Herní služba může nabývat několika různých stavů, které jsou definovány výčtem *GameStatus*. Jedná se o:

GAME_NONE
Žádná hra není načtena. Toto je výchozí stav při vytváření instance služby.

GAME_LOADING
Probíhá načítání scénáře a souvisejících objektů.

GAME_WAITING
Hra je načtena a nyní se čeká na startovní událost. V tuto chvíli jsou aktivní pouze aktualizace polohy (herní časovač neběží), ale nejsou distribuovány scénáři.

GAME_RUNNING
Hra byla spuštěna a vše běží.

GAME_PAUSED
Hra byla pozastavena a nyní se čeká na startovní událost. V tuto chvíli se nepřijímají ani aktualizace polohy, ani neběží herní časovač.

GAME_WON
Hra je ukončena. Hráč vyhrál.

GAME_LOST
Hra je ukončena. Hráč prohrál.

5.6 Skenování kódů

O skenování kódů (QR i jiných) se stará externí aplikace, kterou musí mít uživatel nainstalovanou. V opačném případě se zobrazí dialog s možností jejího stažení a nainstalování. V aktuální implementaci se používá aplikace *QR Droid*[2], ale v případě potřeby je možné ji jednoduše změnit na kteroukoliv jinou (úpravou hodnot ve třídě *Constants*).

Pro aktivaci čtení kódu je nutné zavolat metodu *startScanner()*, která jako parametr přijímá *Activity*, ve které musí být implementována metoda *onActivityResult()* pro předání výsledku skenování zpět herní službě (zavoláním její metody *onActivityResult()*). Ta jej zpracuje a v případě úspěchu rozešle událost *SCANNED_CODE* obsahující naskenovaný kód.

5.7 Použití frameworku

Pro vytvoření aplikace založené na tomto frameworku je potřeba:

1. Vložit knihovnu do projektu a nastavit její použití ve vlastnostech projektu.
2. Vytvořit herní scénář ve formátu XML a vložit ho například do složky *assets*.
3. Vytvořit herní službu poděděním třídy *GameService*, přidat ji do *Manifestu* a naimplementovat požadované metody:

[2]QR Droid je k nalezení na http://qrdroid.com

- **loadScenario()**: Pokud se scénář nenachází ve složce *assets* aplikace, ale je umístěn jinde, je nutné přepsat tuto metodu vlastní implementací. V opačném případě není nutné tuto metodu znovu implementovat.
- **onGameStart()**: Tato metoda je volána po vytvoření služby a úspěšném načtení scénáře. Zde je možné spustit vlastní herní *Aktivity* a podobně.

  Metoda je však volána i při opakovaném požadavku spuštění služby (i když již běží). V tom případě se služba znovu nevytváří a parametr *starting* obsahuje hodnotu *false*.
- **onEvent()**: Tato metoda se volá při každém výskytu herní události. Je vhodné zde kontrolovat speciální herní události a v případě ukončovací události vyhodnotit výsledky a ukončit herní službu.
- **getGameNotification()**: Tato metoda vrací objekt typu *Notification* pro zobrazení oznámení, které je viditelné a průběžně aktualizované během trvání celé hry. Vyžádat aktualizaci (tedy zavolání této metody frameworkem) je možné zavoláním GameService.refreshNotification().

4. Spuštění hry pak probíhá spuštěním herní služby voláním *startService()*. V *Intent* objektu je také nutné předat cestu k souboru se scénářem, který má být spuštěn. Například takto:

```
1 Intent intent = new Intent(this, GameService.class);
2 intent.putExtra("filename", "scenario.xml");
3 startService(intent);
```

5. V *manifestu* aplikace je potřeba specifikovat požadované funkce frameworku (vibrace a přístup k poloze):

```
1 <uses-permission
2   android:name="android.permission.VIBRATE" />
3 <uses-permission
4   android:name="android.permission.ACCESS_FINE_LOCATION" />
```

5.8 Testování

Jednoduché testování tříd bylo prováděno průběžně s psaním jejich kódu. Větší důraz byl poté kladen na otestování celkové funkčnosti a vzájemné komunikace herních objektů. Toho bylo dosaženo vytvořením testovací aplikace, která zároveň demonstruje hlavní funkce frameworku. Program byl otestován jak na emulovaném systému, tak na reálném zařízení.

5.8.1 Testovací aplikace

Pro účely testování byla vytvořena ukázková aplikace, která demonstruje různé možnosti využití frameworku. Po jejím spuštění se zobrazí obrazovka s jedním tlačítkem, po jehož stisknutí se aktivuje herní služba, načte se ukázkový scénář a přepne se na obrazovku s herní mapou. Následně je možné přepínat mezi pěti herními obrazovkami. Obrázek 5.2 ukazuje obrazovku s mapou, pomůckami a systémové oznámení s aktuálním stavem hry.

Mapa

Pro zobrazení mapy je využito *Google Maps Android API v2*[3]. Na mapě jsou pak vykresleny obrysy a plochy herních oblastí a také ikona označující aktuální polohu hráče. Tu ovlivňuje jednak reálný pohyb uživatele, ale její změna se dá také vynutit dlouhým stisknutím na jakékoliv místo na mapě. V obou případech je následně vyvolána herní událost změny pozice hráče a všechny reakce s tím související. Tímto se dá jednoduše otestovat korektní chování hry (co se týče polohy hráče).

Zprávy

Na této obrazovce je seznam herních zpráv. Po kliknutí na zprávu je otevřena obrazovka s jejím detailem a celým textem.

Úkoly

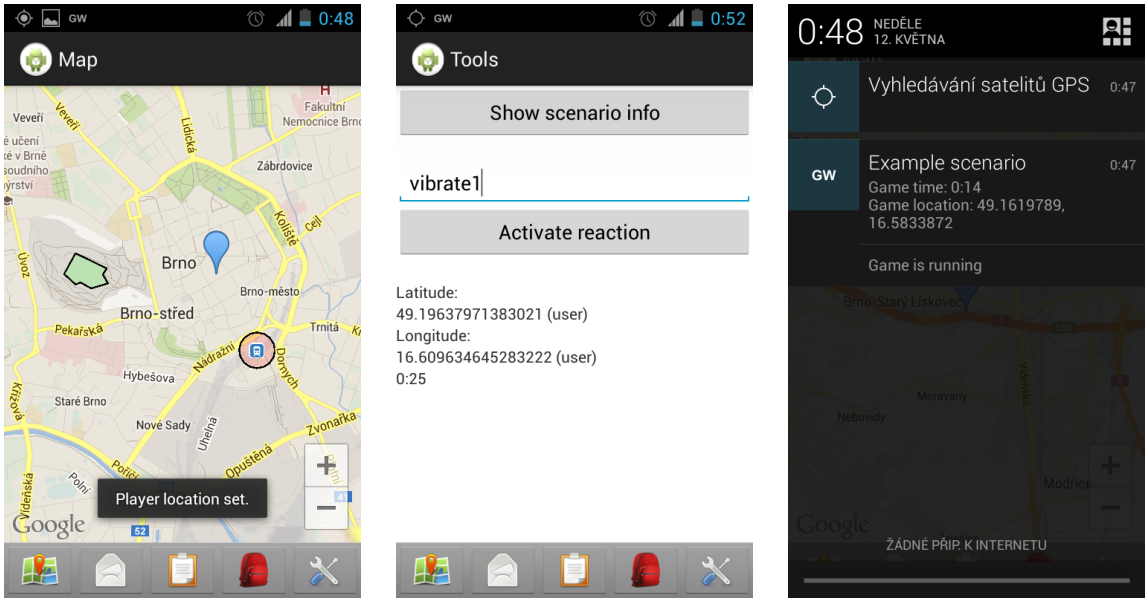
Obrazovka obsahuje seznam herních úkolů. Ve scénáři jsou tyto úkoly definované jako zprávy, ale s odlišným štítkem, proto mohou mít i odlišné využití. V tomto případě se po kliknutí na položku nestane nic.

Inventář

Zde je pouze jedno tlačítko, po jehož stisknutí se zobrazí aplikace pro čtení kódů. Po naskenování kódu se zobrazí zpátky herní aplikace s informací, že kód byl načten.

Pomůcky

Obrazovka s pomůckami umožňuje zobrazit dialog s informacemi o aktuálním scénáři. Také je zde možnost zadat identifikátor *reakce* ze scénáře a ručně ji aktivovat. Toto umožňuje otestování vzájemného chování herních objektů. Dále je zde zobrazen herní čas a GPS souřadnice aktuální polohy hráče (ať už získané senzorem nebo zvolením na mapě).



Obrázek 5.2: Náhledy obrazovek testovací aplikace.

[3]Google Maps Android API v2 lze nalézt na https://developers.google.com/maps/documentation/android/

Kapitola 6

Závěr

Cílem práce bylo navrhnout a implementovat herní engine pro geolokační hry s podporou rozšířené reality. Tyto požadavky byly splněny. Byl vytvořen framework, který umožňuje jednoduchou tvorbu geolokačních her na platformě Android. Společně s ním byla vytvořena i ukázková aplikace, která slouží k demonstraci jeho možností a zároveň usnadňuje i testování funkčnosti konkrétní hry. Díky oddělení definice hry od zbytku kódu a uživatelského rozhraní je dosaženo větší univerzálnosti a znovupoužitelnosti.

K ovládání hry se využívá reálný pohyb uživatele a také jeho přímá interakce s herními obrazovkami (závisí na konkrétní implementaci výsledné aplikace). Propojení s reálným světem je kromě sledování polohy hráče dosaženo možností skenování QR (i jiných) kódů pomocí fotoaparátu zařízení. Hra následně poskytuje uživateli zpětnou vazbu ve formě vibrací, zvuků, změn obrazovek a podobně.

Framework byl zpřístupněn široké veřejnosti na portálu GitHub[1] pod licencí *Apache License Version 2.0*.

Projekt je v aktuálním stavu dostatečně použitelný, ale tento typ her nabízí mnoho oblastí, na které by mohl být zaměřen jeho budoucí vývoj. Rozšířením by mohla být podpora herních entit s umělou inteligencí či možnost propojení více hráčů přes internet. Co se týče herního scénáře, tak i přesto, že je soubor ve formátu XML jednoduše čitelný, jeho úprava může být časově náročná. Z toho důvodu by bylo vhodné vytvořit specializovaný editor, který by práci se soubory scénáře zjednodušoval.

[1] Framework je k nalezení na https://github.com/robyer/gamework

Literatura

[1] ARToolkit: Webová stránka. 2013, [Online; navštíveno 29. 04. 2013].
URL http://www.hitl.washington.edu/artoolkit/

[2] bitstars: DroidAR. 2013, [Online; navštíveno 29. 04. 2013].
URL http://bitstars.github.io/droidar/

[3] Foursquare: Webová stránka. 2013, [Online; navštíveno 29. 04. 2013].
URL https://foursquare.com

[4] Google: Activities. 2013, [Online; navštíveno 29. 04. 2013].
URL http://developer.android.com/guide/components/activities.html

[5] Google: The AndroidManifest.xml File. 2013, [Online; navštíveno 29. 04. 2013].
URL
http://developer.android.com/guide/topics/manifest/manifest-intro.html

[6] Google: Application Fundamentals. 2013, [Online; navštíveno 29. 04. 2013].
URL http://developer.android.com/guide/components/fundamentals.html

[7] Google: Developer Tools. 2013, [Online; navštíveno 29. 04. 2013].
URL http://developer.android.com/tools/index.html

[8] Google: Managing Projects. 2013, [Online; navštíveno 29. 04. 2013].
URL http://developer.android.com/tools/projects/index.html

[9] Google: OpenGL. 2013, [Online; navštíveno 29. 04. 2013].
URL http://developer.android.com/guide/topics/graphics/opengl.html

[10] Google: Sensors Overview. 2013, [Online; navštíveno 29. 04. 2013].
URL
http://developer.android.com/guide/topics/sensors/sensors_overview.html

[11] Google: Services. 2013, [Online; navštíveno 29. 04. 2013].
URL http://developer.android.com/guide/components/services.html

[12] Google: XmlPullParser. 2013, [Online; navštíveno 29. 04. 2013].
URL http:
//developer.android.com/reference/org/xmlpull/v1/XmlPullParser.html

[13] Ingress: Webová stránka. 2013, [Online; navštíveno 29. 04. 2013].
URL http://www.ingress.com

[14] McLaughlin, B.: *Java & XML*. Beijing Cambridge Mass: O'Reilly, 2001, ISBN 0596001975.

[15] Milette, G.: *Professional Android sensor programming*. Indianapolis: John Wiley & Sons, 2012, ISBN 978-1-1181-8348-9.

[16] TrailHit: Webová stránka. 2013, [Online; navštíveno 29. 04. 2013].
URL http://www.trailhit.com

[17] Wikipedie: Geocaching — Wikipedie: Otevřená encyklopedie. 2013, [Online; navštíveno 29. 04. 2013].
URL http://cs.wikipedia.org/wiki/Geocaching

[18] Wikipedie: Geolokace — Wikipedie: Otevřená encyklopedie. 2013, [Online; navštíveno 29. 04. 2013].
URL http://cs.wikipedia.org/wiki/Geolokace

[19] Wikipedie: Zeměpisné souřadnice — Wikipedie: Otevřená encyklopedie. 2013, [Online; navštíveno 29. 04. 2013].
URL http://cs.wikipedia.org/wiki/Zeměpisné_souřadnice

Příloha A

Obsah CD

- **doc*** – programová dokumentace (Javadoc)
- **latex*** – zdrojové kódy technické zprávy
- **src\Gamework*** – zdrojové kódy frameworku
- **src\ExampleGame*** – zdrojové kódy ukázkové aplikace
- **example.xml** – soubor s ukázkovým scénářem a popisem jeho možných atributů
- **readme.txt** – návod na přeložení zdrojových kódů
- **zprava.pdf** – tato technická zpráva v elektornické podobě

Příloha B

Ukázka definice scénáře ve formátu XML

```
<?xml version="1.0" encoding="UTF-8"?>
<scenario>
  <!-- Basic info about scenario (all just string). This element MUST be first element in
  <scenario>. -->
  <about>
    <title>Example scenario</title>   <!-- Name of scenario -->
    <author>Robert Posel</author>   <!-- Name of author -->
    <version>0.0.1 - 5.5.2013</version>  <!-- Version of this game -->
    <location>Brno</location>    <!-- Location where game is played -->
    <duration>1h</duration>     <!-- Estimated duration of one game -->
    <difficulty>Easy</difficulty>   <!-- Difficulty of game -->
    <description>Testing game</description> <!-- Description of game -->
  </about>

  <!-- Definition of game messages. Only <message> elements are allowed inside. -->
  <messages>
    <!-- <message> contains these attributes:
      - id = identificator, must be unique among <messages>
      - title = title of message
      - value = path to file with content of message or short text of message itself
      - tag (not required) = specifies type of message for grouping use - then from scenario you can
      get all messages with same tag
      - default = if set to "true" then this message is visible from start of game automatically
     -->
    <message id="welcome" default="true" title="Welcome to game!" value="messages/test.txt" />
    <message id="test" title="Some test message" value="messages/test.txt" />
    <message id="goal1" tag="goals" default="true" title="Win this game" value="Go to the Veveri and
    then to the Railroad station until time is out..." />
  </messages>

  <!-- Definition of game areas. Only <area> elements are allowed inside. -->
  <areas>
    <!-- <area> contains these attributes:
      - id = identificator, must be unique among <areas>
      - type = one of: point, multipoint, sound
      - other fields depends on particular type

      Types:
      1. point - represents simple circle area with one center point and radius.
      2. multipoint - represents polygon composited from 3 or more points.
      3. sound - is same as Point, but contains source of sound, whose volume changes as user is
      being closer from center point.

      ========================================
      <area type="point"> contains in adition these attributes:
      - lat = latitude of geo location, e.g. 12.3456789
```

```
    - lon = longitude of geo location, e.g. 123.456789
    - radius = radius of area in meters

    <area type="sound"> contains in adition these attributes:
    - lat = latitude of geo location, e.g. 12.3456789
    - lon = longitude of geo location, e.g. 123.456789
    - radius = radius of area in meters
    - soundRadius = radius in which sound will be played, in meters
    - value = path to file with sound sample

    <area type="multipoint"> contains in adition nested <point> elements (at least 3):
      - <point> contains these attributes:
        - lat = latitude of geo location, e.g. 12.3456789
        - lon = longitude of geo location, e.g. 123.456789
   -->
  <area id="railway" type="point" lat="49.1905819" lon="16.6128022" radius="100" />
  <area id="veveri" type="multipoint">
    <point lat="49.193896" lon="16.598848" />
    <point lat="49.193999" lon="16.599222" />
    <point lat="49.193700" lon="16.600036" />
    <point lat="49.193826" lon="16.600658" />
    <point lat="49.193787" lon="16.600849" />
    <point lat="49.193952" lon="16.601334" />
    <point lat="49.194398" lon="16.600937" />
    <point lat="49.194709" lon="16.601235" />
    <point lat="49.195470" lon="16.598830" />
    <point lat="49.194421" lon="16.597960" />
    <point lat="49.193905" lon="16.598794" />
  </area>
</areas>

<!-- Definition of game variables. Only <variable> elements are allowed inside. -->
<variables>
  <!-- <variable> contains these attributes:
    - id = identificator, must be unique among <variables>
    - type = one of: decimal, boolean
    - other fields depends on particular type

    Types:
    1. decimal - represents 32-bit integer variable
    2. boolean - represents boolean variable

    ========================================
    <variable type="decimal"> contains in adition these attributes:
    - value = numeric value of this variable
    - min (not required) = minimal value this variable can hold
    - max (not required) = maximal value this variable can hold
    Note: either both or none min/max values must be set.

    <variable type="boolean"> contains in adition these attributes:
    - value = one of: true, false
   -->
  <variable id="health" type="decimal" min="0" max="100" value="100" />
  <variable id="score" type="decimal" value="0" />
  <variable id="completed" type="boolean" value="false" />
</variables>

<!-- Definition of game reactions. Only <reaction> elements are allowed inside. -->
<reactions>
  <!-- <reaction> contains these attributes:
    - id = identificator, must be unique among <reactions>
    - type = one of: multi, sound, vibrate, message, event, activity, var_set, var_increment,
    var_decrement, var_multiply, var_divide, var_negate
    - other fields depends on particular type

    Types:
    1. multi - represents container containing more <reaction>s
    2. ref - represents reference to another reaction - this makes sense only inside multi reaction.
```

```
  3. sound - plays sound sample
  4. vibrate - vibrates for defined amount of time
  5. message - marks <message> as received (= visible for the player)
  6. event - send one of game events to start, win or lose game
  7. activity - starts any activity defined by it's name
  8. var_set - set (=) value of <variable> (could be numeric or true/false value, depending on
  type of variable)
  9. var_increment - increment (+) value of decimal (only!) <variable>
  10. var_decrement - decrement (-) value of decimal (only!) <variable>
  11. var_multiply - multiply (*) value of decimal (only!) <variable>
  12. var_divide - divide (/) value of decimal (only!) <variable>
  13. var_negate - negate (!) value of boolean (only!) <variable>

  ========================================
  <reaction type="multi"> contains in adition nested <reaction> elements of other types.
  Note: This reaction CAN'T be nested inside another multi reaction.
      Nested reactions DON'T have id and can't be called separately.

  <reaction type="ref"> contains in adition these attributes:
  - value = identificator of another reaction which should be activated

  <reaction type="sound"> contains in adition these attributes:
  - value = path to file with sound sample

  <reaction type="vibrate"> contains in adition these attributes:
  - value = number of ms to vibrate or vibration pattern (e.g. "0, 80, 100, 500")

  <reaction type="message"> contains in adition these attributes:
  - value = id of <message> to be marked as received

  <reaction type="event"> contains in adition these attributes:
  - value = one of GAME_START, GAME_WIN, GAME_LOSE for internal game events, or any other string
  for CUSTOM event

  <reaction type="activity"> contains in adition these attributes:
  - value = full class name (with namespace) of activity which should be started
  Note: id of this reaction is sent to activity through intent's extra string with name "reaction"

  <reaction type="var_*"> contains in adition these attributes:
  - variable = id of <variable> whose value will be changed
  - value = value for operation with variable, depends on reaction type (var_*):
  Note: var_set can change any variable (just use correct value - true/false for boolean
  variables or any integer for decimal variables)
      var_increment, var_decrement, var_multiply, var_divide can be applied to decimal variables
      ONLY.
      var_negate can be applied to boolean variables ONLY.
 -->
<reaction id="alert" type="sound" value="sounds/alert.mp3" />
<reaction id="vibrate1" type="vibrate" value="100" />
<reaction id="vibrate2" type="vibrate" value="500" />
<reaction id="vibrate3" type="vibrate" value="0, 80, 100, 80, 100, 500" />
<reaction id="testmsg" type="message" value="test" />

<reaction id="score10" type="var_increment" variable="score" value="10" />
<reaction id="score50" type="var_increment" variable="score" value="50" />

<reaction id="health_hurt" type="multi">
  <reaction type="var_decrement" variable="health" value="20" />
  <reaction type="ref" value="alert" />
</reaction>

<reaction id="complete" type="multi">
  <reaction type="sound" value="sounds/complete.mp3" />
  <reaction type="vibrate" value="100" />
  <reaction type="var_set" variable="completed" value="true" />
</reaction>

<reaction id="game_win" type="multi">
```

```
    <reaction type="ref" value="vibrate3" />
    <reaction type="sound" value="sounds/win.mp3" />
    <reaction type="event" value="GAME_WIN" />
  </reaction>

  <reaction id="game_lose" type="multi">
    <reaction type="vibrate" value="1400" />
    <reaction type="sound" value="sounds/lose.mp3" />
    <reaction type="event" value="GAME_LOSE" />
  </reaction>

  <reaction id="myev" type="event" value="myevent" />

  <reaction id="help" type="activity" value="cz.robyer.gw_example.activity.HelpActivity" />

  <reaction id="tick_second" type="sound" value="sounds/tick.mp3" />
  <reaction id="tick_minute" type="sound" value="sounds/tick2.mp3" />
</reactions>


<!-- Definition of hooks. Only <hook> elements are allowed inside. -->
<hooks>
  <!-- <hook> contains one or more nested <trigger> elements and these attributes:
    - type = one of: area, area_enter, area_leave, variable, time, event, scanner
    - value = depends on particular type:
      area, area_enter, area_enter: id of area
      variable: id of variable
      time: one of: second, minute, hour
      event: value of your custom event
      scanner: scanned value from code
    Types:
    1. area - this is called when user enters or leaves particular area
    2. area_enter - same as above, but only when enters
    3. area_leave - same as above, but only when leaves
    4. variable - this is called when value of particular variable is changed
    5. time - this is called when time of game changes
    6. event - this is called when is received your custom event
    7. scanner - this is called when user scan a code (QR or other)

    ========================================

    Each <trigger> contains zero or more nested <condition> elements and these attributes:
    - reaction = id of reaction which will be activated
    - conditions (not required) = one of: all, any, none
      all = all conditions must apply in order to activate reaction
      any = at least one condition must apply in order to activate reaction
      none (or no conditions attribute) = reaction is activated instantly without conditions
    - run (not required) = number of allowed runs of this trigger. Not set or value <= 0 means
    without limits.

    ========================================

    Each <condition> contains these attributes:
    - type = one of: equals, notequals, greater, smaller, greaterequals, smallerequals
    - value = number or true/false value to be compared (depends on type of variable which will be
    compared)
    - variable = id of variable whose value will be compared (not required)
    Note: variable is not required only for these <hook> types: variable, time. In this case will
    be compared value of condition with value of changed variable/time defined in hook.
   -->
  <hook type="variable" value="health">
    <trigger reaction="game_lose" conditions="ALL">
      <condition type="equals" value="0" />
    </trigger>
  </hook>

  <hook type="area_enter" value="veveri">
    <trigger reaction="complete" run="1" />
```

```
    </hook>

    <hook type="area_enter" value="railway">
      <trigger reaction="game_win" conditions="ALL">
        <condition type="equals" variable="completed" value="true" />
      </trigger>
    </hook>

    <hook type="time" value="second">
      <trigger reaction="tick_second" conditions="ALL" />

      <trigger reaction="health_hurt" conditions="ANY">
        <condition type="equals" value="0" />
        <condition type="equals" value="20" />
        <condition type="equals" value="40" />
      </trigger>
    </hook>
    <hook type="time" value="minute">
      <trigger reaction="tick_minute" conditions="ALL">
        <condition type="greater" value="0" />
      </trigger>
      <trigger reaction="game_lose" conditions="ALL">
        <condition type="greater" value="5" />
      </trigger>
    </hook>
    <hook type="scanner" value="8594026360061">
      <trigger reaction="myev" />
    </hook>
    <hook type="event" value="myevent">
      <trigger reaction="game_win" />
    </hook>
  </hooks>

</scenario>
```