VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

WEBOS KLIENT PRO SYSTÉM WHERIGO

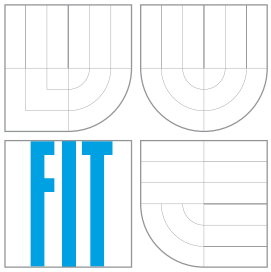
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB JELEN

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

WEBOS KLIENT PRO SYSTÉM WHERIGO

WEBOS CLIENT FOR WHERIGO

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAKUB JELEN

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. TOMÁŠ KAŠPÁREK

BRNO 2013

Abstrakt

Moje práce je zaměřena na návrh hybridních aplikací pro platformu webOS a na studium struktury systému Wherigo z programátorského hlediska. Jsou zde řešeny problémy s kompatibilitou aplikací na různých zařízeních webOS. Dále také problémy spojené s neexistencí oficiální dokumentace Wherigo ze strany tvůrce. Výsledkem je funkční a otestovaná aplikace pro Wherigo dostupná v katalogu aplikací webOS a splňující požadavky zadání.

Abstract

In my thesis I focus on the design of hybrid application for webOS platform and on study and understanding of structure of the Wherigo system from programmer's point of view. I deal with compatibility problems between webOS devices. Another issue is an insufficiency of documentation available for the Wherigo library. The result is fully functional and tested application for Wherigo that meets the requirements and is available in the webOS App Catalog.

Klíčová slova

webOS, Wherigo, Javascript, Enyo framework, Lua, Geocaching, embedded Lua, Lua C API, GPS

Keywords

webOS, Wherigo, Javascript, Enyo framework, Lua, Geocaching, embedded Lua, Lua C API, GPS

Citace

Jakub Jelen: WebOS klient pro systém Wherigo, bakalářská práce, Brno, FIT VUT v Brně, 2013

WebOS klient pro systém Wherigo

Prohlášení

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

.....
Jakub Jelen
13. května 2013

Poděkování

Děkuji Tomáši Kašpárkovi za vedení práce a cenné rady. Dále děkuji Janu Matějkovi za úvod do problematiky tvorby Wherigo přehrávače a doporučení, která jsem od něj dostal. Také bych chtěl poděkovat skupině Wherigo Foundation za tipy a znalosti, které bych bez její podpory jen těžko získával. V neposlední řadě také děkuji Martinu Jílkovi a Michalovi Kuchtovi za poskytnutí zdrojových kódu jejich Wherigo cartridgí.

Obsah

Kapitola 1

Úvod

Pro svoji bakalářskou práci jsem zvolil implementaci programu pro mobilní operační systém webOS. V úvodu představím tento operační systém z pohledu uživatelského, jeho vývoj a způsob tvorby aplikací pro tuto platformu. Dále následuje popis platformy Wherigo a souvisejících souborových formátů.

V další kapitole jsou zmíněny požadavky na aplikaci z uživatelského hlediska, navrženo řešení propojení jazyků C++ a Lua, postup návrhu vlastních objektů Wherigo knihovny a analýza použitých typů souborů, které v aplikaci používám.

Další část již obsahuje detaily implementace pro systém webOS, strukturu aplikace s rozdělením na logické a fyzické celky v různých programovacích jazycích a použité algoritmy.

Poslední kapitola shrnuje metody testování a obsahuje příklady cartridgí, na kterých byla moje aplikace otestována, jak v reálné podobě, tak virtuálně, pomocí simulace GPS dat.

1.1 Operační systém webOS

Historie a vývoj

WebOS je operační systém pro mobilní telefony vyvinutý společností Palm, představený začátkem roku 2009 na veletrhu spotřební elektroniky CES[1]. Již při představení přilákal mnoho pozornosti a první telefon s tímto operačním systémem získal několik ocenění [1]. Přes počáteční úspěchy se firma potýkala s mnohými problémy, a tak byla již začátkem roku 2010 i se systémem koupena firmou HP. Ta v systém vkládala velké naděje a měla v plánu používat jej v různých vestavěných systémech. Ale již koncem roku 2011 bylo ohlášeno uvolnění systému jako opensource. Komunita se ujala otevřeného zdrojového kódu a dnes, v roce 2012, existují funkční porty webOS na telefony jiných výrobců. Jméno se změnilo na Open webOS a oddělení se přejmenovalo na GRAM [2]. Začátkem roku 2013 firma HP prodala webOS firmě LG, která má práva využít systém nebo jeho části ve svých chytrých televizích [3].

Technická data

Systém je postaven na linuxovém jádře stejně jako Android, ale uživatelské rozhraní a aplikace jsou tvořeny, jak už název napovídá, webovými technologiemi - javascriptem,

[1] Consumer Electronics Show

HTML a CSS. Aplikace jsou spouštěny v prostředí WebKit. Systém splňuje všechny předpoklady pro použití na chytrém telefonu včetně připojení k WiFi, datovým sítím operátora, GPS přijímače, Bluetooth modulu a standardního 3.5 mm jack audio výstupu. Dalším důležitým předpokladem pro dobré využití chytrého telefonu je dostatek aplikaci, ale o tom se zmíním až v sekci č 1.2.

Operační systém je primárně ovládán přes dotykový displej, jehož dotyková vrstva přesahuje zobrazovací plochu a umožňuje zadávat dotyková gesta (pohyby)[2] pro funkce „zpět“ či otevření launcheru. Až do druhé verze systému se počítalo s druhou vstupní metodou, tedy s přítomností hardwarové QWERTY klávesnice[3]. Ta navazuje na precizní klávesnice původních přístrojů Treo od Palmu, průkopníků smartphonů, a i přes relativně malé rozměry se velmi dobře používá. Ve verzi webOS 3.0, která je určena pro tablety bez hardwarové klávesnice, je vstupem pouze dotykový displej se softwarovou klávesnicí.

I přes neúspěchy na trhu obsahuje systém mnoho pokrokových funkcí, jako třeba nepřekonaný multitasking s použitím karet reprezentujících aplikace nebo jednoduchý systém notifikací, na který někteří uživatelé nedají dopustit. Vývoj všech mobilních operačních systémů jde rychle dopředu a některé z úspěšných funkcí postupně pronikají i do jiných operačních systémů.

1.2 Aplikace

Jak již bylo zmíněno výše, původní koncept byl postavit aplikace pouze na webových technologiích, kterým rozumí skoro každý, a tak by bylo velmi jednoduché získat mnoho aplikací třetích stran. Ze začátku to vypadalo jako velmi dobrý tah, ale toto rozhodnutí s sebou neslo několik, na první pohled trochu skrytých, úskalí:

- Většina zdrojového kódu je v nepříliš oblíbeném Javascriptu.
- Interpretace Javascriptu je pomalá, což bylo dost znát na prvních telefonech.
- Některé věci v Javascriptu udělat nelze (programování blíže hardware – i když Palm vytvořil rozhraní pro komunikaci javascriptu se službami operačního systému).

I přesto vzniklo mnoho zajímavých aplikací pro tuto platformu. Ze začátku byly do tohoto systému vkládány naděje k záchraně oslabující společnosti Palm, ale firma v té době neměla rychlé dodavatele a nezvládla distribuci do celého světa. Jelikož trh s mobilními telefony jde velmi rychle dopředu, dnes (jaro 2013), po čtyřech letech, existuje jen malá komunita uživatelů, kteří tento systém používají a kteří pro něj vyvíjejí aplikace. Celkový počet aplikací je v řádech tisíců [3].

Dokumentace k vývoji aplikací obsahuje podrobný popis životního cyklu aplikace, který pomáhá vývojáři zorientovat se v mnoha možnostech spouštění aplikace a v běhu na pozadí. Také přibližuje přístup ke službám systému. Dále obsahuje seznam bodů, které má dobrá aplikace splňovat, a to jak z hlediska vzhledového, tak co se týče chování a práce s pamětí [4].

Frameworky

Pro tvorbu aplikací přišel Palm s Mojo frameworkem, který obsahuje widgety uživatelského rozhraní a základní strukturu aplikace. Widgety uživatelského rozhraní jsou ve formě

[2] Z angličtiny Touch gestures – gesta, Swipes – pohyby, tahy
[3] Rozložení se liší podle cílového trhu

HTML části pro umístění do DOM[4] a javascriptové inicializace. Toto použití je docela rozvláčné a kvůli tomu s příchodem nových verzí od HP vznikl nový framework Enyo, který tyto dvě části spojil a umístil do objektu v Javascriptu. Ten určuje zanořením strukturu a zároveň zajišťuje generování DOM a inicializaci jednotlivých elementů.

Samotný Mojo Framework podporuje tvorbu aplikací podle architektonického vzoru MVC[5]. Podle toho jsou také pojmenované výchozí adresáře ve struktuře aplikace.

Nativní aplikace

Jak jsem již výše naznačil, Javascript není všemocný. To si brzy uvědomili i inženýři z Palmu. Dalším logickým krokem bylo umožnit vyvíjet nativní aplikace. Nativním částem se v prostředí webOS říká „plugin“. K javascriptovému SDK bylo přidáno PDK[6]. Tyto aplikace lze psát v C/C++, využívat SDL[7] pro grafické aplikace a pomocí dalších rozhraní komunikovat se zbytkem systému. Díky tomu bylo na webOS portováno několik her, včetně celosvětově oblíbených Angry Birds [5].

Hybridní aplikace

Tyto dva přístupy lze do jisté míry kombinovat a těžit z výhod obou. Jedná se o uživatelské rozhraní napsané v jednom z výše zmíněných frameworků, tudíž vzhledově nevybočuje z řady výchozích aplikací a podporuje koncepty dobré aplikace [4]. Druhou částí aplikace je potom plugin, což je spustitelný soubor napsaný v C/C++, zkompilovaný pro architekturu ARM, se kterým může aplikace obousměrně komunikovat. Tento plugin může provádět výpočetně náročnější úkoly nebo obsahovat části již napsané v tomto jazyce pro jiné platformy.

1.3 Wherigo

Wherigo je hra pro zařízení s GPS, která přenáší virtuální dobrodružství do reálného světa. V přístroji se odehrává virtuální svět, který je ovládán pomocí GPS a často interaguje s reálným okolím. V základu se jedná o rozšíření Geocachingu, ale lze vytvořit i nezávislé turistické průvodce nebo modifikace různých her (Whack a mole, Battleship, Snakes and ladders).

Historie

Systém byl vyvinut společností Groundspeak v roce 2008, primárně pro GPS přístroje značky Garmin a pro kapesní počítače s operačním systémem Pocket PC (Microsoft) [6]. Brzy si získal mnoho uživatelů a začaly vznikat zajímavé hry. Samotný Groundspeak od vydání prvních verzí mnoho nového neudělal, ale komunita brzy začala tvořit vlastní přehrávače[8] a editory[9]. Ze editorů bych zmínil pouze české URWIGO [7], ale existujícím přehrávačům se budu věnovat o trochu více, protože to bude náplní mé práce.

[4] Document Object Model
[5] Model - View - Controller
[6] Plugin Development Kit
[7] Simple DirectMedia Layer
[8] častěji používané anglicky „player“
[9] z anglického „builder“ lze těžko přeložit

Přehrávače

Přehrávač Wherigo cartridge má za úkol provést hráče fyzickými místy a dodat mu prvky virtuální reality na displeji zařízení. Může uživateli prezentovat virtuální objekty, stopovat čas a požadovat po uživateli interakci formou dialogů a dotazů.

Hlavní obrazovka se skládá ze čtyř částí:

Locations - Místa, kam se může hráč vydat

You See - Věci, které hráč vidí v okolí

Inventory - Věci, které má hráč virtuálně u sebe

Tasks - Úkoly, které má hráč splnit, nebo je již splnil

Obrázek 1.1: Screenshot oficiálního Wherigo Emulatoru pro Windows s otevřenou catridge „O třech prasátkách“. Část popisů uživatelského rozhraní lze počeštit.

Oficiální přehrávače Groundspeak společně s projektem Wherigo uvedl přehrávače pro platformu Pocket PC a pro turistické navigace společnosti Garmin. Tyto přehrávače jsou brány jako referenční, protože jsou stále nejvyužívanější.

openWIG Jako první neoficiální přehrávač vznikl projekt openWIG od autora Matejcik [8] (Jan Matějek) z pražské UK. Na přehrávači začal pracovat již v roce 2008 a dokázal vytvořit kompatibilní program bez jakékoliv dokumentace a podpory Groundspeaku. Přehrávač je implementován v Javě, primárně pro mobilní telefony (J2ME), ale knihovnu využil také pro vytvoření desktop verze.

WhereYouGo Projekt Wherigo přehrávače pro Android vznikl v roce 2010 a autorem je Jiří Mlavec z ČVUT [9]. Aplikace využívá Wherigo knihovnu z openWIG. Tímto počinem se dostalo Wherigo na velmi rozšířenou platformu.

PiGo, později Wherigo Aplikace napsaná pro širokou základnu telefonů, pro přístroje Apple. Konkrétního autora se mi nepodařilo dohledat, ale tato aplikace je vytvořena s podporou Groundspeaku.

xmarksthespot Implementace Wherigo ve skriptovacím jazyce python pro snadnou přenositelnost a spustitelnost na různých platformách. Aplikace je stále ve vývoji, ale zdrojový kód [10] mi byl v mnoha místech inspirací. Autorem je Bas Wijnen z Německa.

Kapitola 2

Analýza aplikace a návrh

Úkol vytvořit přehrávač Wherigo cartridgí s sebou nese mnoho úskalí, stejně jako přenos aplikace na tuto platformu. V první části vás seznámím se samotnou hrou, se specifickými vlastnostmi platformy a problémy, které je potřeba řešit.

2.1 Wherigo cartridge

Základním stavebním prvkem, který v této aplikaci od uživatele dostanu, je právě Wherigo cartridge – binární soubor, unikátní pro každého uživatele, který lze stáhnout z portálu Wherigo.com a který obsahuje samotnou hru. Konkrétně tedy hlavičku s metadaty cartridge, hlavní zkompilovaný bytekód jazyka Lua a nakonec další soubory (obrázky a zvuky) související se hrou. Formát cartridge není nikde oficiálně zveřejněn, ale byl zdokumentován tvůrci aplikací na jiné platformy. Pro přehlednost je zde také umístěn jako příloha A.

Soubor je třeba číst s ohledem na binární obsah a respektovat použité datové typy. Jednotlivé soubory je třeba extrahovat a umístit do adresáře, který bude přístupný z uživatelského rozhraní, ale neměl by být pro uživatele viditelný z ostatních aplikací (prohlížeč fotek, hudební přehrávač).

2.2 Databáze her

Pro rychlý přehled o uložených cartridgích v telefonu je dobré mít jednoduchou databázi s hlavičkami extrahovanými z cartridgí. Měl by v ní být příznak o rozbalení cartridge (extrakce souborů do datového adresáře aplikace), o aktuálním stavu hry (uložení) a o případném dokončení. Po odstranění cartridge z databáze musí být odstraněny také extrahované soubory z paměti telefonu, aby nedocházelo k plnění souborového systému.

Uložené cartridge by mělo být možné filtrovat podle typu z hlavičky (Geocache, TourGuide, Puzzle, Fiction) a podle stavu hry (příznak uložená, dokončená).

Cartridge by se měly automaticky řadit podle vzdálenosti od aktuální pozice. Speciálním případem jsou cartridge typu „Play Anywhere“, které musí být umístěny ve speciálním seznamu, protože nemají zadanou výchozí pozici.

Databázi je potřeba aktualizovat na vyžádání uživatele nebo jako reakci na událost smazání, uložení nebo dokončení cartridge.

2.3 Komunikace se serverem wherigo.com

Výhodou dnešních chytrých telefonů oproti jednoúčelovým navigacím je připojení k internetu, které je dostupné již téměř všude. Aplikace by tedy měla mít možnost komunikovat se serverem, autorizovat uživatele, vyhledávat on-line nejbližší cartridge, prohlížet zápisy ostatních uživatelů, stahovat do paměti telefonu cartridge a po dokončení umožnit uživateli cartridge odemknout na webu a napsat zážitky z hry.

Aktuálně (jaro 2013) však portál wherigo.com nemá žádné veřejné API, přes které by s ním mohly aplikace komunikovat, a proto se ve své práci tomuto bodu věnovat nebudu. Tento bod je zde zařazen pouze pro budoucí možnost rozšíření aplikace.

2.4 Uživatelské rozhraní

Pro tvorbu uživatelského rozhraní mám několik možností. Je možné použít jeden z dodávaných frameworků:

Mojo Starší, funguje na všech starších přístrojích, ale není již podporován systémem openWebOS, vývoj v něm je poněkud neohrabaný, jak jsem si vyzkoušel při tvorbě své první aplikace před několika lety.

Enyo Nový framework, kompatibilní s novějšími přístroji, na starší jde jednoduše doinstalovat, tvorba aplikací v něm je výrazně jednodušší, protože se učí z chyb prvního frameworku.

Teoretickou třetí možností by bylo implementovat uživatelské rozhraní přímo v C++ za pomocí jiného frameworku, ale zde by byly jasné nevýhody v uživatelské přívětivosti (jiný vzhled než ostatní aplikace) a jádro problému, který řeším, by se pravděpodobně přesunulo na implementaci uživatelského rozhraní.

Po zvážení těchto ohledů je nejvhodnější možnost použít moderní Enyo framework, který navíc nabízí spustitelnost na jiných platformách.

2.5 Lua

Ze souboru cartridge získáme zkompilovaný zdrojový kód Lua (Lua Bytekód), který stačí interpretovat. Jako interpret je nejjednodušší zvolit oficiální implementaci ze stránky lua.org, která je velice dobře dokumentovaná a otestovaná. Jedná se o zdrojové kódy v C, které lze zkompilovat pro cílovou architekturu mobilního telefonu a připojit k samostatné aplikaci.

Pro spuštění bytekódu z cartridge je potřeba použít verzi 5.1 namísto aktuální 5.2, protože bytekód není kompatibilní mezi verzemi.

Knihovna Wherigo

V dalším kroku je třeba vyřešit tento problém: zdrojové kódy her vyžadují (direktiva *require*) knihovnu Wherigo, ale tato knihovna není nikde veřejně publikovaná.

Pro přehled o objektech, funkcích a konstantách jsem musel nastudovat soubory generované oficiální aplikací Wherigo Builder, které poskytují základní strukturu knihovny

Wherigo. Některé cartridge využívají nestandardní konstrukce, které je potřeba vypozorovat a aplikaci pro ně odladit. Dalším dobrým zdrojem jsou ostatní opensource implementace Wherigo, které jsou zmíněny v sekci 1.3.

Tuto knihovnu je potřeba zpřístupnit z prostředí Lua VM. To lze udělat dvěma způsoby, které se od sebe velmi liší:

- Třídy definovat v C++ a exportovat do prostředí, ve kterém budeme hry spouštět. Tento způsob by přinesl jednodušší přístup k datům cartridge z C++, ale mnoho práce s těsným propojením těchto dvou jazyků.
- Třídy definovat v Lua, z C++ exportovat pouze funkce ovládající UI, systémové akce a další věci, které nelze v Lua zpracovat. Takto je implementován emulátor a pravděpodobně také oficiální přehrávače. Toto řešení má také výhodu, že není kvůli každé změně v knihovně potřeba znovu kompilovat celý projekt.

Objekty a funkce, které je třeba implementovat shrnuje příloha C, podrobněji jsou popsány na přiloženém CD (generováno z komentářů pomocí LuaDoc).

Propojení s C++

Propojení jazyka Lua s C++ lze řešit několika způsoby:

- Provést export dat z C++ (informace v hlavičce cartridge, nativní funkce) ručně pomocí funkcí Lua C API a ručně také zpracovávat parametry funkcí a návratové hodnoty.
- Provést export dat pomocí nějaké knihovny k tomu určené, což přinese jednodušší práci s funkcemi, ale potenciálně nižší výkon.

Rozhodl jsem se se využít knihovnu LuaBridge [11], která je distribuována pod licencí MIT a která by dokázala zpracovávat i objekty. Já ji používám pouze pro funkce a proměnné ve jmenných prostorech Env a WIGInternal, které jsou cartridgemi vyžadovány.

Události a synchronizace

Aplikace je událostmi řízená. Mezi nejčastější události patří změna pozice, při které je potřeba přepočítat všechny aktivní zóny a vzdálenosti. Další důležité události jsou start, uložení, ukončení a obnovení cartridge ze souboru. Dále jsou to akce herních objektů zobrazených uživateli. Tyto události mohou ovlivňovat zobrazení v uživatelském rozhraní.

Události hry (zobrazení zprávy) jsou neblokující a synchronní (zobrazí se ihned, uživatelské rozhraní běží v nezávislém procesu). Události uživatelského rozhraní (klik na položku v inventáři) jsou zachyceny funkcemi ve vlastních vláknech a je třeba zajistit jejich synchronní zpracování.

Objekty v Lua

Programovací jazyk Lua nemá vestavěný koncept tříd. Objekty lze simulovat pomocí tabulek, jak je předvedeno v článku na serveru Lua users.org [12], ze kterého při implementaci vycházím.

Vzhledem k tomu, že nevytvářím vlastní knihovnu, ale napodobuji již existující pro požadavky stávajících zdrojových kódů, je potřeba přizpůsobit například názvy konstruktorů, které nejsou standardizované.

Objekt je simulován typem tabulka, který má podobné vlastnosti – může obsahovat proměnné a funkce. Konstruktor objektu je potřeba simulovat přiřazením metatabulky s metametodou `__call`, která umožní tabulku volat jako funkci a umožní vrátit nový „objekt“, tedy vlastně opět tabulku. Destruktor v jazyce Lua nemusíme řešit, protože Lua obsahuje Garbage collector a nepotřebné objekty odstraňuje samovolně. Třídy jsou vytvořeny na základě techniky prototypů, podobně jako v jazyce Javascript.

Součástí jazyka je zkrácená forma zápisu metod s automatickým předáváním instance objektu. Ta nám umožňuje zapisovat `obj:func(p)` na místo `obj.func(obj,p)` při definici i volání objektových metod.

Tabulky objektu musí být nějak označeny, abychom později poznali, že se jedná o objekt, a abychom dokázali určit, ke které třídě tato instance patří. Pro tento případ je dobré přidat do tabulky položku s nekolizním názvem (například `_classname`) a vytvořit třídám funkce `made`, které ověří příslušnost objektu k dané třídě.

Metatabulky

Některé objekty v knihovně Wherigo se umějí chovat jako čísla (objekty *Distance* a *Bearing*) a lze je přímo porovnávat s jinými čísly, nebo jinými objekty. K tomuto účelu existují v metatabulkách další klíče, které využívám, konkrétně `__eq`, `__lt`, `__le` pro porovnávání a `__add`, `__sub`, `__mul`, `__div` pro použitelné aritmetické operace. Bohužel, nelze definovat funkci převodu na číselný datový typ (tak, jako je pro převod na řetězec použita funkce `__tostring` z metatabulky) a je třeba definovat všechny tyto možnosti pro korektní chování některých cartridgí, které této možnosti využívají.

Metatabulky dále využívám pro sledování hodnot některých vlastností, které je třeba zaznamenávat, nebo které mohou vyvolat událost, například `OnSetActive`. Používám klíče `__index` a `__newindex`, které jsou volány, pokud je požadován neexistující index pro čtení nebo zápis.

2.6 Uložené hry

Možnost uložení hry je součástí originálního systému. Neoficiální přehrávače řeší tento problém různě, a proto zde mám několik možností:

- Následovat standard a ukládat hru do souboru kompatibilního s oficiálními přehrávači, což vyžaduje zdokumentování struktury souboru, ale umožní odemykání hry tímto souborem na webu wherigo.com.
- Provést export celého běhového prostředí virtuálního stroje Lua[1], což by bylo datově velmi objemné. Na druhou stranu by byla jistota úspěšného a kompletního obnovení všech, i nestandardních konstrukcí.
- Během hry sledovat změny v objektech globálního prostředí a zaznamenávat pouze tyto změny. Toto řešení by vyžadovalo speciální metatabulky sledující změny vlastností za běhu a ve výsledku bych stejně musel ukládat do nějakého formátu podobného první možnosti. Současně by celý proces hraní byl výpočetně více náročný, i kdyby uživatel hru nakonec neukládal.

[1] Lua Virtual Machine

Formát uložených her

Formát používaný emulátorem a oficiálními přehrávači není nikde popsaný, ale lze jej odhadnout analýzou souboru generovaného emulátorem. Mnoho částí jsou řetězcové úseky (jména vlastností, objektů, řetězcové konstanty), které jsou přímo čitelné, u dalších částí definujících strukturu lze odvodit jejich význam podle kontextu.

I přesto, že vycházím z původní implementace Wherigo, nevyužívám některé vlastnosti využívané oficiálními přehrávači. Pro export je potřeba tyto vlastnosti vygenerovat a pro načítání je zpracovávat do struktury, kterou používám.

Časování ukládání hry

Problém může nastat, když vyžadujeme uložení hry inicializované funkcí cartridge. V tuto chvíli by bylo třeba ukládat ještě zásobník volání a hodnoty. Tomuto se lze docela jednoduše vyhnout způsobem, který je naznačen pojmenováním metody objektu cartridge: `RequestSync` neznamená okamžitou synchronizaci (=uložení), ale požadavek na uložení, jakmile skončí aktuální strom volání funkcí.

Takového chování lze jednoduše docílit vložením uživatelské SDL události[2] do zásobníku událostí ve chvíli požadavku. Po dokončení předchozího požadavku se již provede uložení z hlavní programové smyčky, protože lua je volána v jediném procesu a nevytváří žádná vlákna (v terminologii Lua: *coroutine*).

[2] SDL user event

Kapitola 3

Implementace a spuštění na cílové platformě

3.1 Instalace software pro vývoj

Prerekvizitou instalace je oficiální vydání Oracle Java Runtime Environment (JRE). Ze stránek pro vývojáře HP/Palm je třeba dále stáhnout SDK/PDK instalátor. Pro tvorbu spustitelných souborů na ARM architektuře je vyžadován CodeSourcery Toolchain. Tento balík je podporován pouze operačními systémy Windows a Mac OS. Pro instalaci emulátoru je potřeba mít nainstalovaný Virtualbox [13].

3.2 Příprava zařízení pro vývoj

Aktivace vývojářského módu

Vývojářský mód lze, na rozdíl od ostatních platforem, odemknout velmi jednoduše. Do univerzálního vyhledávacího pole stačí zadat text `webos20090606` nebo Konami kód `upupdowndownleftrightleftrightbastart` a ve zobrazené aplikaci „Developer Mode Enabler“ přepněte přepínač do polohy On.

Po restartu je přístroj spuštěn v režimu, který umožňuje instalovat aplikace z počítače a přistupovat k chybovým a ladícím výpisům aplikací.

Příkazová řádka v přístroji

Pro přístup k příkazové řádce v přístroji je třeba nainstalovat do telefonu SSH démona, který bude komunikovat přes USB. To lze provést připojením zařízení k počítači a spuštěním utility z nainstalovaného vývojového kitu:

```
$ pdk-device-install install
```

Po restartu získáme přístup přes `putty` (Windows) nebo `novaterm` (Linux, součástí instalace SDK) k příkazové řádce, kde můžeme spouštět a ladit aplikace přímo na zařízení, například pomocí `gdb` [14].

Enyo framework pro starší přístroje

WebOS verze 1.x a 2.x obsahuje aplikace používající framework Mojo. Pro spuštění aplikace využívající Enyo framerowk je potřeba ho doinstalovat, a to buď společně s aplikací „Maps“ z oficiálního katalogu, nebo jako samostatný balík „Enyo 1“ z Preware.

Hybridní aplikace pro starší přístroje

Starší přístroje obsahují chybu, a proto aplikace napsaná podle tutoriálů nekomunikuje s pluginem. Jediná zmínka o řešení je ve zdrojových kódech Enyo[1]: Kromě popsaných postupů stačí odeslat událost `__PDL_PluginStatusChange__` s parametrem „ready“ po dokončení registrace ostatních metod[2].

3.3 Podpůrné nástroje

HP podporuje pro vývoj PDK pouze operační systém Windows a Mac, a proto překlad probíhal právě ve virtualizovných Windows, které jsou pro mě dostupnější.

Makefile

Pro sestavení převzatých zdrojových kódů interpretu Lua a vlastního C++ kódu je GNU Makefile nejjednodušší řešení. V OS Windows je třeba nastavit proměnnou PATH pro aplikace GNU. V dodaném Makefile sestavujícím knihovny Lua už stačí nahradit původní nástroje za nástroje z Code Sourcery a provést křížový překlad[3].

Makefile obsahuje cíle pro překlad, zabalení aplikace do *.ipk balíčku, nebo okamžité spuštění na připojeném zařízení. Dále obsahuje přepínač pro překlad a spuštění na lokálním systému pro ladění chyb, úniků paměti a další testování. Dalším přepínačem je odstranění ladících symbolů pro publikaci v oficiálním katalogu.

Vycházel jsem ze vzorové implementace na oficiálním vývojářském webu [15], tudíž je možné zvolit přepínače překladu specifické pro různá zařízení, nebo zvolit univerzální pro všechny platformy s webOS.

Funkčnost uživatelského rozhraní

Uživatelské rozhraní aplikace je psáno v javascriptu, za pomocí frameworku Enyo, a tudíž je možné jeho funkčnost ověřit nezávisle na platformě. Stačí k tomu webový prohlížeč s jádrem WebKit. Já využívám otevřený prohlížeč Chromium. Prohlížeč je potřeba spustit s parametry, které umožní spouštět lokálně uložené skripty a ukládat cookie do počítače uživatele [16]:

```
chromium-browser --disable-web-security --enable-file-cookies
```

V prohlížeči je poté spuštěno celé uživatelské rozhraní, algoritmy lze ladit pomocí vestavěných ladicích nástrojů a kontrolovat chyby. Hledání chyb v javascriptu je tak mnohem rychlejší a jednodušší než aplikaci pokaždé instalovat na zařízení, spouštět a v systémovém logu hledat chyby.

[1] `<https://github.com/enyojs/enyo-1.0/blob/master/framework/source/palm/controls/Hybrid.js\#L49>`

[2] Viz. zdroj Engine.cpp, řádky 595 - 602

[3] Z anglického výrazu „Cross compile“

Lze simulovat také události specifické pro zařízení, podobně jako v Emulátoru. „Pohyb zpět“[4] lze simulovat klávesou Escape, otevření nabídky aplikace pomocí kombinace kláves `Ctrl + ‘`.

Pomocí speciálních funkcí (`enyo.nextTick()` pro simulaci asynchronních odpovědí) a proměnných (`window.PalmSystem` pro detekci spuštění v zařízení) lze simulovat odpovědi hybridní části aplikace pro ověření funkčnosti částí závislých na nativní části.

Analýza souborů s uloženou hrou

Pro jednodušší analýzu a porovnání souborů s uloženými hrami jsem vytvořil jednoduchou aplikaci převádějící soubor GWS (binární formát) do formátu XML, který je lépe čitelný a hlavně porovnatelný pomocí nástrojů příkazové řádky.

Program je pojmenován `gws2xml` a lze sestavit na Linuxu cílem `gws2xml` programu make. Očekává pouze jeden parametr – jméno GWS souboru. Výsledné XML je posíláno na standardní výstup, který lze jednoduše přesměrovat do souboru. Porovnání lze provést standardními nástroji `diff`. Pro případ hledání rozdílných polí v uloženém souboru je užitečné soubory řádky seřadit nástrojem `sort`, například takto:

```
$ vimdiff <(gws2xml cart.gws.webos | sort) <(gws2xml cart.gws.emu | sort)
```

Pro zajímavost takto generovaný XML soubor je menší, než binární formát GWS.

luadec

Při spouštění bytekódu v prostředí Lua nelze získat dostatečně podrobné chybové výpisy. Přestože jsem se snažil pracovat s opensource cartridgemi, občas jsem narazil na problém u cartridge, ke které jsem neměl k dispozici zdrojový kód. Pro odhalení speciálních konstrukcí používaných tvůrci cartridgí naštěstí existuje projekt `luadec` [17], který převádí bytekód opět na zdrojový kód Lua. Transformace není dokonalá, výsledný kód často nejde znovu spustit, ale pro nalezení nestandardních konstrukcí je to dostatečné.

Přeložení a použití je jednoduché, jak je popsáno na výše citované stránce. Pro vstup je potřeba použít soubory *.luac z pracovního adresáře aplikace v telefonu nebo v počítači. Aplikace vyžaduje lua knihovnu a spuštění na kompatibilní platformě vůči cartridge (32bitový operační systém).

```
$ luadec cartridge.luac > cartridge.lua
```

3.4 Struktura aplikace

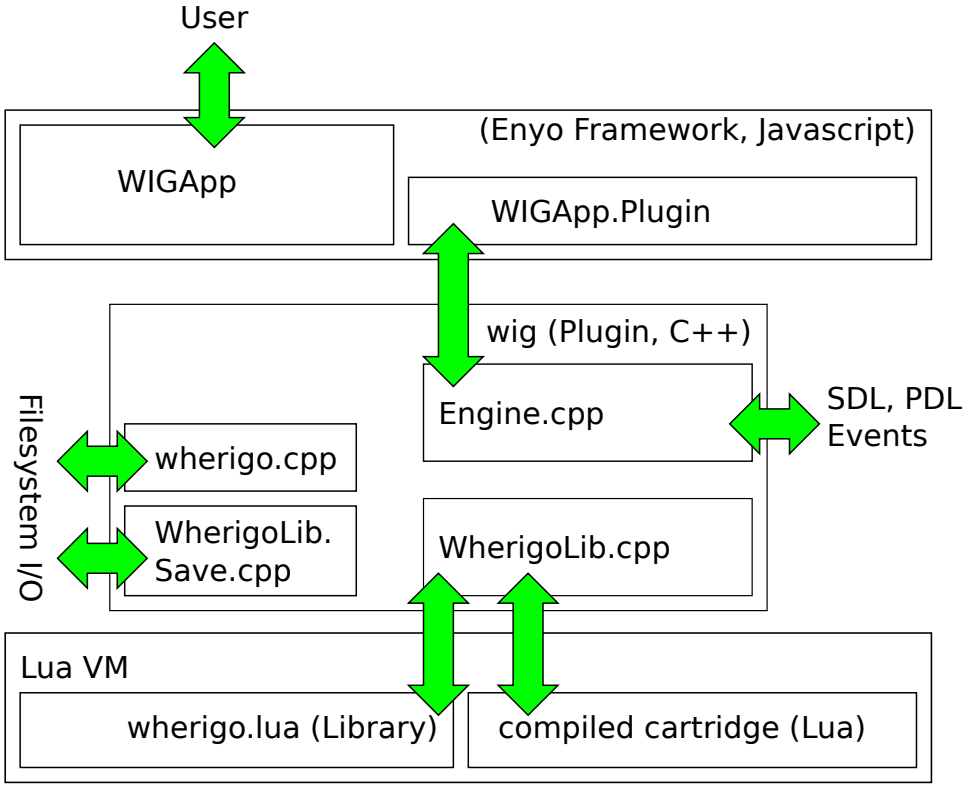
Na obrázku č. 3.1 jsou naznačeny datové toky mezi jednotlivými částmi aplikace. V následujícím textu se pokusím popsat jednotlivé logické celky.

Uživatelské rozhraní – Enyo framework

Uživatelské rozhraní je postaveno na hlavním objektu `WIGApp`, který obsahuje selektor obrazovek[5] hlavní nabídky – umožňuje zobrazení seznamu cartridgí, detailu cartridge, nastavení aplikace a zobrazení samotné hry. Soubory s objekty pro jednotlivé obrazovky jsou umístěny ve složce `enyo`.

[4] z anglického „Back swipe“

[5] v terminologii Enyo: Pane - umožňuje zobrazení různých typů obsahu a přechody mezi nimi



Obrázek 3.1: Schéma aplikace a datových toků mezi jednotlivými částmi.

Obrazovka samotné hry obsahuje, pro kontrolu opuštění hry, další výběr obrazovek zobrazitelných ve hře – celkový přehled, detail kategorie a obrazovku s detailem položky.

Speciálním objektem je `Plugin`, který zajišťuje komunikaci s nativní částí aplikace. Další objekty řeší zobrazení různých vyskakovacích oken a požadavků na vstup uživatele a výpočet vzdáleností GPS souřadnic pro zobrazení seznamu nejbližších cartridgí.

Navigace v aplikaci Uživatel se v aplikaci pohybuje pomocí stisku tlačítek, nebo výběrem různých položek v seznamu. Tímto způsobem je řešeno zanořování do různých selektorů obrazovek[6]. Postupné vynořování je řešeno pomocí události „Back“, která je vyvolána tahem vlevo v ploše pro gesta[7]. WebOS verze 3.x a Open webOS již nezaručuje přítomnost plochy pro gesta, a proto je na každé obrazovce také tlačítko umožňující vrátit se zpět. Celkový pohled na strukturu obrazovek popisuje obrázek č. 3.2.

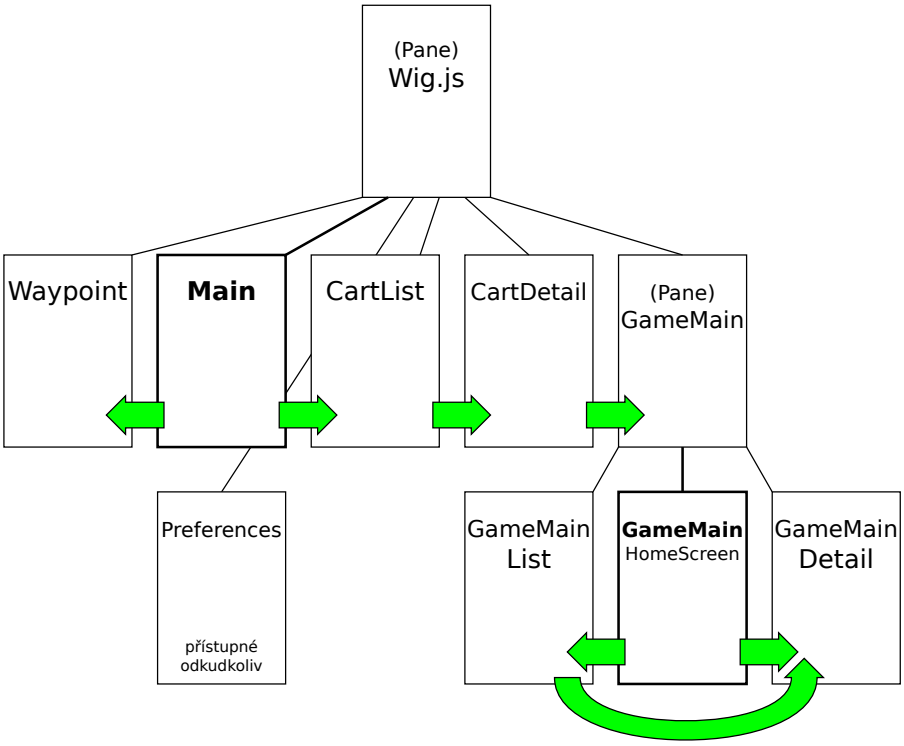
Plugin – C++ aplikace

Jedná se o jakési jádro celého projektu, i když může vypadat pouze jako most spojující jednotlivé části. Pro přehlednost je zdrojový kód rozdělen na logické celky do jmenných prostorů, které se starají o různé úkoly.

Jmenný prostor `Engine` obstarává komunikaci s uživatelským rozhraním, systémem a předávání jeho událostí do jmenného prostoru `WherigoLib`. Ten se stará o zpracová-

[6] V názvosloví frameworku Enyo se jim říká „Pane“

[7] A anglického „Gesture area“



Obrázek 3.2: Schéma obrazovek v uživatelském rozhraní. Tučně jsou zobrazeny výchozí obrazovky selektorů a šipky znázorňují možnost zanořování. Speciální obrazovkou je obrazovka „Preferences“, která je přístupná z jakékoliv jiné obrazovky přes aplikační menu.

vání všech událostí a správu běhového prostředí Lua pro hru. Součástí je jmenný prostor `WherigoLib::Save`, který řeší ukládání hry do souboru a její obnovu.

Další významnou částí je objekt `Wherigo`, který zpracovává samotný GWC soubor, načítá do svých vlastností parametry hlavičky a zpřístupňuje samostatnou hru mezi jmennými prostory.

Mezi pomocné objekty lze zařadit objekty `FileReader` a `FileWriter`, což jsou objekty nad vstupním nebo výstupním proudem a zpřístupňují metody pro čtení různě dlouhých úseků a jejich korektní převod na datové typy využité v C++, viz tabulky použitých datových typů v přílohách A a B.

Knihovna Wherigo – Lua

Knihovna byla vytvořena na základě originální knihovny používané v emulátoru a některé algoritmy jsou inspirované verzí pro Python [10]. Jedná se o samostatný zdrojový soubor `wherigo.lua`. Kromě vyžadovaných objektů, funkcí a konstant (viz sekce 2.5 a příloha C) obsahuje také funkce provádějící výpočty, které by bylo složité dělat přes C API knihovny Lua. Především zpracování požadavků a odpovědí uživatelského rozhraní.

3.5 Komunikace mezi částmi aplikace

Komunikační kanály mezi Enyo frameworkem a C++

První rozhraní, které aplikace obsahuje, je mezi uživatelským rozhraním v Javascripitu a pluginem v jazyce C++.

WebOS má pro tento problém vlastní knihovnu a funkce, které tuto komunikaci umožňují a zjednodušují jak na straně javascriptu třídou `enyo.Hybrid`, tak na straně C++ hlavičkovým souborem `PDL.h`. Ten mimo jiné definuje způsob komunikace, potřebné metody a konstanty. Požadavky pro hybridní aplikaci lze shrnout do těchto bodů:

- V javascriptu (Enyo framework):
  - Vytvořit novou třídu (kind) `enyo.Hybrid` a určit spustitelný soubor, se kterým se bude komunikovat.
  - Určit metody, které lze z pluginu volat.
  - Implementovat zpracování těchto volání.
- V C++ pluginu:
  - Provést inicializaci PDL: `PDL_Init(0);`
  - Provést inicializaci SDL: `SDL_Init(SDL_INIT_VIDEO);`
  - Registrovat obslužné funkce pro přijímání volání z Enyo frameworku.
  - Oznámit frameworku dokončení inicializace (viz poznámka o starších zařízeních v sekci 3.2).
  - Přijetí parametrů funkcí z Javascriptu a alokace paměti. Toto je prováděno ve vlastním vlákně, a proto je potřeba předat parametry SDL událostí.
  - Zpracování události v hlavním vlákně a uvolnění paměti.
  - Zakódování návratové hodnoty do JSON, pokud je třeba.
- V appinfo.json zveřejnit položku `"plug-ins": true`.
- Do výsledného *.ipk balíčku přidat:
  - spustitelný soubor plugin (zkompilovaný pro architekturu ARM).
  - soubor package.properties obsahující nastavení atributů spustitelnosti souboru plugin (`filemode.755=plugin`).
  - soubor APP_NAME_plugin_appinfo.json (kde APP_NAME je jméno aplikace z appinfo.json) obsahující informace o spustitelné aplikaci (nepovinné).

Vybrané komunikační kanály v mé aplikaci

getCartridges(refresh) Požadavek UI a odpověď obsahující seznam cartridgí v datovém adresáři. Odpověď může vzniknout bez závislosti na požadavku.

openCartridge(filename) Požadavek UI na otevření cartridge `filename` a asynchronní potvrzující odpověď.

save Požadavek UI na uložení hry.

ClosePrompt Požadavek cartridge na uzavření hry.

closeCartridge(save) Požadavek UI na uzavření právě otevřené cartridge, parametr `save` značí, zda se má hra uložit, nebo ne.

deleteCartridge(filename) Požadavek UI na odstranění souboru cartridge a všech ostatních dočasných souborů uložených v souborovém systému telefonu.

MessageBox(message, media, button1, button2, callback) Požadavek cartridge na zobrazení zprávy s textem `message`, obrázkem `media`, tlačítky `button1`, `button2` a pokud vyžaduje odpověď, je nastavena hodnota `callback` na true a je odeslána odpověď závislá na vybraném tlačítku.

playAudio(media) Požadavek cartridge na spuštění přehrávání zvuku `media`.

GetInput(type, text, choices, media) Požadavek cartridge na vstup uživatele. Hodnota `type` může nabývat hodnot Text, Number, MultipleChoice. Položka `text` obsahuje otázku zobrazenou uživateli, `choices` je pole možností pro typ MultipleChoice a `media` je zobrazený obrázek. Po potvrzení uživatelem je odeslána asynchronní odpověď.

showScreen(screen, item) Požadavek cartridge na zobrazení určité obrazovky. Pokud se jedná o detail, `item` určuje unikátní identifikátor objektu na dané obrazovce.

showMap(zone_id) Požadavek UI na zobrazení mapy okolí se všemi body zóny a asynchronní odpověď.

updateState(JSONdata) Pravidelná aktualizace stavu otevřené cartridge do UI. Obsahuje všechna data, která jsou zobrazena uživateli.

CallbackFunction(event, ObjId) Požadavek UI pro vykonání události (zvolený příkaz, nebo OnClick událost).

switchGPS(newState) Požadavek UI na zapnutí nebo vypnutí automatické aktualizace GPS pozice. Vyžadováno pro testování a může ušetřit baterii při delších přesunech bez vypnutí aplikace.

setPosition(lat, lon) Požadavek UI na nastavení pozice. Pro ladicí a testovací účely umožňuje nastavit pozici uživatele a simulovat vstup GPS.

movePosition(delta_lat, delta_lon) Požadavek UI na posun pozice uživatele pro ladicí účely.

Komunikační kanály mezi C++ pluginem a knihovnou Lua

Další rozhraní, které moje aplikace vyžaduje, je provázání pluginu v C++ s třídami v Lua. Pro propojení funkcí jsem použil koncept podobný Wherigo emulátoru, a to implementaci knihovny v jazyce Lua a export pouze funkcí, které vyžadují speciální zpracování, nebo požadavek pro uživatelské rozhraní. Tyto funkce jsou sdruženy do jmenného prostoru `WIGInternal`, který používají také oficiální přehrávače.

Volání C++ funkcí ze strany Lua je řešeno pomocí knihovny LuaBridge.

Rozhraní je podobné výše zmiňovanému, ale jsou zde navíc funkce časovače. Funkce popsané výše jsou již vynechány.

Ve směru k C++

escapeJsonString Provede ošetření znaků, které nelze přímo serializovat do řetězce JSON. Pro vyšší výkon implementováno v C++

RequestSync Požadavek cartridge na uložení rozehrané hry - vytvoří SDL událost, která je vykonána po dokončení aktuálně prováděného zpracování v programové smyčce. Tímto je ošetřen problém ukládání zásobníku volání.

Close Požadavek na uzavření cartridge, vyžaduje potvrzení uživatele

addTimer(remaining, ObjId) Vytvoří nový časovač na `remaining` sekund. `ObjId` je unikátní ID časovače v cartridge.

removeTimer(ObjId) Zruší běžící časovač, zadaný podle `ObjId`

getTime Vrací aktuální čas. Pro výpočet zbývajícího času pro opakující se události.

LogMessage(message) Posílá zprávu do logu průběhu cartridge

Ve směru k Lua Volání Lua funkcí ze strany C++ je řešeno přes zásobník Lua a funkce C API knihovny. Funkce, které jsou interní, jsou zařazeny také do jmenného prostoru Wherigo knihovny, ale začínají podtržítkem, aby se snížilo riziko kolize s uživatelem definovanými funkcemi a proměnnými.

ZTimer._Tick(ObjId) Událost vypršení časovače

Wherigo._MessageBoxResponse(value) Reakce uživatele na zobrazení zprávy (vybrané tlačítko)

Wherigo._GetInputResponse(value) Odpověď uživatele na otázku

Wherigo._callback(event, ObjId) Událost na objektu (Click, Command)

Wherigo._getUI Žádost o data pro zobrazení v uživatelském rozhraní

ZCartridge._update(position, time, accuracy) Aktualizace pozice GPS, času a přesnosti

Další přístupy C++ do prostředí Lua a do objektů vlastní hry jsou řešené přes Lua C API. Takto je řešeno získání statických dat hry, která jsou odesílána jako odpověď funkci `openCartridge`, události OnStart a OnRestore, nebo získání dat pro zobrazení mapy. Stejným způsobem je řešen přístup do prostředí Lua při ukládání/obnovování hry ze souboru.

3.6 Algoritmy

Sférická geometrie

Pro práci s GPS souřadnicemi je potřeba využít algoritmy sférické geometrie. Země sice není přesná koule, ale odchylky od ní jsou minimální a zanedbatelné. Je potřeba počítat s extrémními případy kolem severního pólu a s přechodem kolem datové hranice.

Je bod v zóně? Mezi velmi často používané algoritmy patří ověření, zda je hráč uvnitř zóny. Jakýkoliv uzavřený rovinný útvar, jimž zóna definována alespoň třemi body je, dělí kulovou plochu na dvě části a pro korektní výpočet by bylo vyžadováno, aby bylo určeno, jaká část je uvnitř a jaká venku. Zdrojové soubory her obsahují vlastnost OriginalPoint – jakýsi střed zóny. Ale tento bod není vždy uvnitř zóny (například pro nekonvexní tvary). Kromě toho Emulátory a ostatní přehrávače tento bod ignorují. V rámci kompatibility s existujícími hrami mi nezbývá, než se zachovat stejně.

Základním algoritmem, který pro tento úkol používám, je algoritmus Inpoly [18], který využívá metodu „počtu přechodů“[8]. Hlavní myšlenkou je ověřit, kolikrát se polopřímka vedená z ověřovaného bodu na sever protne se všemi segmenty zóny. Pokud je tento počet lichý, jsme v zóně, jinak jsme mimo ni. Algoritmus popsaný ve výše uvedeném článku používá několik optimalizací pro hraniční případy.

Algoritmus bez dalších optimalizací nelze použít pro výpočet zón, které obsahují severní nebo jižní pól, případně které protínají datovou hranici. Tento stav je velmi nepravděpodobný, a proto jsem jej zanedbal. Stejně tak v tomto algoritmu považuji zemi za plochou, protože většina zón je relativně malá a v tomto případě je odchylka zanedbatelná.

Dále jsem použil optimalizaci pomocí MBR[9], která ještě před spuštěním výše uvedeného algoritmu ověří, zda jsme vůbec v blízkosti této zóny.

Vzdálenost a azimut Výpočet nejkratší vzdálenosti a azimutu na kulové ploše není úplně zřejmý, a proto bych jej chtěl zde vysvětlit. Vycházel jsem z článku [19], který čitelně popisuje problémy a poskytuje vhodné algoritmy.

Výpočet vzdálenosti mezi dvěma body lze řešit několika různými způsoby, ale studium každé metody, výběr nejvhodnější, popřípadě tvorba vlastní metody by zdaleka přesáhly rozsah této práce. Já jsem zvolil metodu „Haversine formula“ (nenalezl jsem český překlad), která je využita i ve výše uvedeném článku a je definována takto:

$$R = 6371 \text{ [km]} \tag{3.1}$$

$$a = \sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right) \tag{3.2}$$

$$c = 2\text{atan2}\left(\sqrt{a}, \sqrt{1-a}\right) \tag{3.3}$$

$$d = Rc \tag{3.4}$$

Kde R je poloměr Země, φ je zeměpisná šířka a λ je zeměpisná délka. Důležité je dosazovat úhly/souřadnice v radiánech.

Algoritmus pro výpočet azimutu v sobě skrývá také několik překvapení. Prvním je fakt, že azimut se po cestě z jednoho bodu do druhého po nejkratší cestě mění. Pro naše potřeby stačí určit azimut ve výchozím bodě, protože ten nás povede po nejkratší cestě. Ten je definován takto:

$$\theta = \text{atan2}\left(\sin(\Delta\lambda)\cos(\varphi_2), \cos(\varphi_1)\sin(\varphi_2) - \sin(\varphi_1)\cos(\varphi_2)\cos(\Delta\lambda)\right) \tag{3.5}$$

Vycházíme opět ze zeměpisné šířky a délky φ a λ v radiánech. Výsledný azimut je opět v radiánech.

[8] z anglického „Crossing Count“

[9] Minimal Bounding Rectangle, speciální případ Bounding box pro dva rozměry

Za zmínku možná ještě stojí funkce atan2, což je běžná trigonometrická funkce Arkus tangens se dvěma parametry pro určení správného kvadrantu na základě znamének předaných argumentů.

Posun bodu o danou vzdálenost v daném azimutu Další algoritmus, který je použitý v knihovně a v uživatelském rozhraní, je algoritmus offsetu bodu. Vycházel jsem z výše zmíněného zdroje [19]. Použité vztahy vypadají takto:

$$\varphi_2 = \text{asin}(\sin(\varphi_1)\cos(d/R) + \cos(\varphi_1)\sin(d/R)cos(\theta)) \tag{3.6}$$

$$\lambda_2 = \lambda_1 + \text{atan2}(\sin(\theta)\sin(d/R)\cos(\varphi_1), \cos(d/R) - \sin(\varphi_1)\sin(\varphi_2)) \tag{3.7}$$

Kde R je opět poloměr Země, φ je zeměpisná šířka, λ je zeměpisná délka, d je vzdálenost a θ je požadovaný azimut.

Ukládání a obnova rozehraných her

Originální Wherigo přehrávače podporují ukládání a obnovování her, což je součástí většiny rozsáhlých her. Pro ukládání souborů využívá Emulátor a Garmin formát GWS, jehož struktura je přiložena jako příloha B. Tuto strukturu jsem se rozhodl respektovat a vytvořil jsem kompatibilní soubor.

Z objektů je potřeba odfiltrovat funkce knihovny Wherigo, které nemá smysl ukládat, a některé vlastnosti, které nejsou v referenčním souboru potřeba. Naopak některé vlastnosti ve své knihovně nepoužívám, a proto je potřeba tyto vlastnosti pro ukládání vytvořit.

Data jsou zapisována ve formátu, v jakém je získávám z Lua C API. Předtím je potřeba přenést proměnné z globálního jmenného prostoru do pole `ZVariables`, aby byly společně se hrou uloženy. Opačně je potřeba postupovat při obnovování ze souboru.

Kapitola 4

Testování

Testování částí aplikace probíhalo od začátku vývoje, a to v několika rovinách. Automatizované testy byly použity pro ověření funkčnosti samostatné knihovny, ale nejdůležitější bylo testování v terénu s reálnými GPS daty a opravdovými cartridgemi.

4.1 Automatizované testy

Aplikace je rozsáhlá a různorodá, a proto se mi nepodařilo pokrýt touto metodou všechny části. Automatizované testy jsem vypracoval pouze pro část Wherigo knihovny. Testy ověřují možné konstrukce používané v různých cartridgích a chování funkcí knihovny při určitých akcích. Soubor testů je uložen ve složce `tests/` a lze spustit zadáním argumentu test, spustitelnému souboru:

```
$ ./wig test
```

Alternativně lze spustit pouze některé testy zadáním jejich jména ve druhém parametru.

4.2 Testy uživatelského rozhraní

Uživatelské rozhraní nelze otestovat automaticky. Testování spočívalo v prohlížeči Chromium a na dvou různých telefonech s různou verzí webOS. Testoval jsem porovnání vůči emulátoru, zda lze provést všechny možné kombinace akcí, zda všechna tlačítka a nastavení reagují. Dále probíhala občasná kontrola vůči jiným aplikacím během terénního testování.

V porovnání s emulátorem jsem se dopustil několika významnějších změn, které vedou k jednoduššímu a rychlejšímu používání, i když mohou někoho zmást.

- Možnost zobrazit detail položky jediným dotykem z hlavní herní obrazovky
- Zobrazení seznamu příkazů položky bez vyžadování stisku dalšího tlačítka

Původní systém byl vytvořen pro jiný druh ovládání a na menší displeje. V dnešní době je na displeji místa dost a rolování obrazovky je velmi intuitivní a rychlé.

4.3 Terénní testy

Asi nejdůležitější částí testování hotové aplikace byly testy v terénu a ověření chování aplikace v prostředí, ve kterém má být využita. Během konečné části vývoje jsem se vypravil na většinu Wherigo cartridgí, které jsou v Brně dostupné, abych ověřil, že je možné jejich bezproblémové dokončení. Dále jsem vyzkoušel několik cartridgí, které je možné hrát kdekoliv[1].

4.4 Simulace GPS

Na všechny cartridge se nemohu dostat, a i kdybych mohl, není nejpohodlnější testovat vždy v terénu. Největší problémy je nejlepší odchytit pomocí simulace u počítače. Během spolupráce s tvůrci Wherigo cartridgí jsem dostal doporučení na velmi rozsáhlou a komplikovanou cartridge, Feuersturm über Reutlingen[2], která využívá téměř všech vlastností Wherigo (jak standardních, tak velmi nestandardních), a proto pro mě byla cennou zkušeností.

Obrázek 4.1: Snímek obrazovky aplikace.

Změna GPS pozice je možná pouze z obrazovky zóny, nebo objektu, který má definovanou pozici, a tudíž se v detailu zobrazí kompas a vzdálenost k cíli, jak je vidět na obrázku č. 4.1.

[1] Například tyto: O třech prasátkách od Siky:
<http://www.wherigo.com/cartridge/details.aspx?CGUID=c885657e-5eab-4958-a1b1-0058e1e48bfb>,
Whack-A-Lackey od Ranger Fox:
<http://www.wherigo.com/cartridge/details.aspx?CGUID=f6002f33-c68a-4966-82ca-3c392a85d892>
Battleship od Ranger Fox:
<http://www.wherigo.com/cartridge/details.aspx?CGUID=9065c3cf-f837-4c50-b418-63a7d9f3b485>

[2] Feuersturm über Reutlingen od Charlenni:
<http://www.wherigo.com/cartridge/details.aspx?CGUID=9573732b-2242-413d-a5d5-b33b8b5ba2b6>

Pozici lze ovládat tlačítkem „Move To“, které okamžitě přesune hráče na pozici zóny nebo věci (velmi nepřirozený skokový pohyb, který může některé cartridge zmást), nebo pomocí kláves (v textovém boxu vpravo dole) `w`, `s`, `d`, `a` pro pohyb po větších krocích a kláves `i`, `k`, `l`, `j` pro pohyb po menších krocích ve směru světových stran (S, J, V, Z), jak je zobrazeno na obrázku č. 4.2[3].

Obrázek 4.2: Simulace GPS pomocí klávesnice

Simulace je v publikované aplikaci skryta, protože není součástí hry a hráče by naváděla k podvádění. Simulaci GPS lze zapnout pomocí konstanty `DEBUG` v souboru `Wig.js`.

[3]Zdroj: `<http://www.productwiki.com/upload/images/hp_pre_3_1.jpg>`

Kapitola 5

Závěr

Výsledkem projektu je funkční aplikace, která je srovnatelná s aplikacemi na jiných platformách a která byla otestována na okolních cartridgích. Plánuji pokračovat ve vývoji a rozšířit uživatelské rozhraní o interakci s webovým portálem `<http://www.wherigo.com>` (viz sekce 2.3), což přinese mnohem lepší uživatelskou přívětivost a dokáže plně využít potenciálu Wherigo her na mobilním zařízení. Aplikace je nyní dostupná v oficiálním katalogu aplikací pod jménem WebWIG [20]. Zdrojové kódy jsou dostupné v repozitáři GitHub[1].

Rozšířil jsem si znalosti o programování aplikací pro mobilní zařízení, naučil jsem se pracovat se skriptovacím jazykem Lua, který lze využít pro jednoduché skripty, vkládat do kompilovaných jazyků a těžit z výhod jejich kombinace. Naučil jsem se objektově orientované programování v Lua a při implementaci vkládání do C++ jsem nahlédl pod pokličku Lua VM. Současně jsem si osvěžil znalosti o C++.

Hlavním problémem, který jsem musel řešit, byla neexistence oficiální dokumentace platformy Wherigo. Často jsem musel dělat pokusy s emulátorem, porovnávávat binární soubory a sledovat chování aplikace v různých mezních situacích. Výsledkem mého hledání jsou přílohy k této práci, které shrnují detaily systému Wherigo, a samotné zdrojové soubory. Dále potom dokumentace samostatné knihovny, která je přiložena na CD. Zjištěné údaje o systému Wherigo mohou být použitelné pro autory přehrávačů na ostatních platformách k zlepšení jejich funkčnosti a kompatibilitiy.

[1] `<https://github.com/Jakuje/webwig>`

Literatura

[1] Wikipedia: Palm Pre. <http://en.wikipedia.org/w/index.php?title=Palm_Pre&oldid=519487141>, 2012 [cit. 2012-11-09].

[2] Open webOS website. <http://www.openwebosproject.org>, 2012 [cit. 2012-11-28].

[3] Wikipedia: WebOS. <http://en.wikipedia.org/w/index.php?title=WebOS&oldid=552059624>, 2013 [cit. 2013-04-22].

[4] Palm: Application Guidelines. <https://developer.palm.com/content/resources/distribute/application_checklist.html>, 2011 [cit. 2013-01-09].

[5] Bohn, D.: First Look: Angry Birds on webOS [Video] Update: Available Now! <http://www.webosnation.com/first-look-angry-birds-webos-video-update-available-now>, 2010 [cit. 2013-04-16].

[6] Groundspeak Inc.: Wherigo website. <http://www.wherigo.com>, 2012 [cit. 2012-11-28].

[7] YourSelf: URWIGO website. <http://urwigo.com/>, 2011 [cit. 2013-04-25].

[8] ZCh: Wherigo po česku : Matejcik. <http://www.wherigo.cz/autori/matejcik/>, 2012 [cit. 2012-11-28].

[9] Mlavec, J.: Android Wherigo client - Google Project Hosting. <http://code.google.com/p/android-whereyougo/>, 2012 [cit. 2012-11-28].

[10] Wijnen, B.: Python module for Wherigo cartridge files. <https://github.com/wijnen/python-wherigo>, 2012 [cit. 2013-03-31].

[11] Falco, V.: LuaBridge 1.0.2. <https://github.com/vinniefalco/LuaBridge>, 2012 [cit. 2013-04-16].

[12] Lua users.org: Object Orientation Closure Approach. <http://lua-users.org/wiki/ObjectOrientationClosureApproach>, 2011 [cit. 2013-04-15].

[13] Palm: Downloading and Installing the 3.0.5 SDK and PDK. <https://developer.palm.com/content/resources/develop/sdk_pdk_download.html>, 2011 [cit. 2013-04-23].

[14] webOS Developer Center, H.: Device Shell. <https://developer.palm.com/content/api/dev-guide/tools/device-shell.html>, 2013 [cit. 2013-04-25].

[15] Unwiredben: A Walk Through the Plug-In Development Kit, Part 3. <http://developer.palm.com/blog/2010/09/a-walk-through-the-plug-in-development-kit-part-3/>, 2010 [cit. 2012-12-21].

[16] Thornton, A.: Extended Enyo Tutorial. <https://developer.palm.com/content/resources/develop/extended_enyo_tutorial.html>, 2011 [cit. 2013-04-05].

[17] VirusCamp: luadec - a lua decompiler based on Hisham Muhammad and SztupY's luadec. <http://code.google.com/p/luadec/>, 2011 [cit. 2013-04-15].

[18] Stein, B.: A Point about Polygons. <http://www.visibone.com/inpoly/>, 1997 [cit. 2013-04-06].

[19] Veness, C.: Calculate distance, bearing and more between Latitude/Longitude points. <http://www.movable-type.co.uk/scripts/latlong.html>, 2012 [cit. 2013-04-15].

[20] Jelen, J.: WebWIG. <https://developer.palm.com/appredirect/?packageid=com.dta3team.app.wherigo>, 2013 [cit. 2013-04-08].

[21] Siegmund, P.: Wherigo cartridge file (.GWC) structure. <http://code.google.com/p/wherugo/wiki/GWC>, 2012 [cit. 2012-11-12].

[22] Matejcik: WherigoBuilder - Globals. <http://wherigobuilder.wikispaces.com/Globals>, 2009 [cit. 2013-04-23].

Příloha A

Struktura souboru Wherigo Cartridge (GWC)

Popis souboru GWC souboru, převzato, přeloženo a doplněno z wiki aplikace WherUGo [21]. Datové typy popisuje tabulka č. A.1

- `@0000`: Identifikace souboru (Magic number):

```
BYTE      0x02
BYTE      0x0a
BYTE      "CART", 0x00
```

- `@0007`: Počet objektů v souboru:

```
USHORT    number_of_objects
```

- `@0009`: Přesně number_of_objects referencí na jednotlivé objekty v cartridge:

```
repeat <number_of_objects> times {
    USHORT object_id   ; Unikátní ID pro každý objekt
    LONG address     ; adresa v souboru
}
```

- `@xxxx`: Hlavička cartridge

```
LONG  header_length     ; Délka hlavičky (následujícího bloku):
struct {
    DOUBLE  Latitude    ; N+/S-
    DOUBLE  Longitude   ; E+/W-
    DOUBLE  Altitude    ; Nadmořská výška
    LONG    unknown1
    LONG    unknown2
    ; media-ID of icon and splashscreen:
    SHORT  ID_of_splashscreen     ; -1 = bez úvodní obrazovky
    SHORT  ID_of_small_icon       ; -1 = bez ikony
    ASCIIZ  type_of_cartridge     ; Typ: např "Wherigo cache"
```

```
    ASCIIZ  PlayerName            ; Jméno hráče
    LONG    unknown6
    LONG    unknown7
    ASCIIZ  CartridgeName         ; Jméno cartridge
    ASCIIZ  CartridgeGUID
    ASCIIZ  CartridgeDescription  ; Popis cartridge
    ASCIIZ  StartingLocationDescription ; Popis výchozího bodu
    ASCIIZ  Version               ; Verze cartridge
    ASCIIZ  Author
    ASCIIZ  Company
    ASCIIZ  RecommendedDevice     ; Doporučené zařízení
    LONG    unknown8
    ASCIIZ  CompletionCode
} header
```

- `@xxxx + header_length + 4`: Zde by měl být první objekt, ale jejich offsety jsou definovány seznamu referencí
- `@address_of_FIRST_object` objekt ID = 0 – vždy bytekód LUA

```
LONG          length
BYTE[length]  content_of_object ; LUA bytekód
```

- `@address_of_ALL_OTHER_objects` objekt ID > 0

```
BYTE  valid_object
if (valid_object == 0) {
    ; Objekt byl smazán a neexistuje v cartridge
} else {
    LONG  object_type  ; Typ objektu: 1=bmp, 2=png, 3=jpg, 4=gif,
                       ; 17=wav, 18=mp3, 19=fdl, ostatní neznámé
    LONG  length
    BYTE[length]  content_of_object ; objekt
}
```

Datový typ	popis
BYTE	unsigned char (1 byte)
SHORT	signed short (2 bytes)
USHORT	unsigned short (2 bytes)
LONG	signed long (4 bytes)
ULONG	unsigned long (4 bytes)
DOUBLE	double-precision floating point number (8 bytes)
ASCIIZ	zero-terminated string (”hello world!”, 0x00)

Tabulka A.1: Datové typy použité pro popis struktury souborů

Příloha B

Struktura souboru Wherigo Savegame (GWS)

Popis struktury souboru s uloženou hrou.

- `@0000`: Identifikace souboru (Magic number):

```
BYTE    0x02
BYTE    0x0a
BYTE    "SAVE", 0x00
```

- `@0007`: Délka hlavičky v bytech

```
LONG    header_length
```

- `@000A`: Jméno cartridge

```
struct {
    ASCIIZ    CartridgeName
    BYTE[8]   CartridgeDateOfCreation
    ASCIIZ    PlayerName
    ASCIIZ    DeviceName
    ASCIIZ    DeviceID      ; S/N zařízení
    BYTE[8]   DateOfSaving
    ASCIIZ    NameOfSave    ; "UI Initialized sync"
    DOUBLE    Latitude
    DOUBLE    Longitude
    DOUBLE    Altitude
} header
```

- `@000A + header_length`: Počet objektů obsažených v souboru

```
LONG    number of ZObjects
```

- `@000A + header_length + 4`: Seznam objektů

```
repeat <number of ZObjects> times {
    LONG            length
    BYTE[length]    ObjectName
}
```

- `@xxxx`: Tabulka objektu Player

```
LONG                length
BYTE[length]        ZCharacter (ObjectName)
OBJECT              Player
```

- `@xxxx`: Přesně number of ZObjects tabulek s ostatními objekty z tabulky AllZObjects

```
repeat <number of ZObjects> times {
    LONG            length
    BYTE[length]    ObjectName
    TABLE           Object
}
```

Použité datové typy z jsou vysvětleny v tabulkách č. A.1 a B.1.

Datový typ	binární formát	popis
BOOL	0x01 + BYTE	00 = false, 01 = true
NUMBER	0x02 + DOUBLE	
STRING	0x03 + LONG(l) + BYTE[l]	Řetězec bez ukončovací nuly
FUNCTION	0x04 + LONG(l) + BYTE[l]	lua_dump() funkce
TABLE	0x05 + (KEY + VALUE)* + 0x06	Formát dvojic: klíč, hodnota
REFERENCE	0x07 + USHORT	Ref. na ZObjekt pomocí ObjId
OBJECT	0x08+LONG(l)+BYTE[l]+TABLE	Jméno objektu

Tabulka B.1: Datové typy použité pro popis struktury souboru

Příloha C

Knihovna Wherigo

Knihovna je uložena ve jmenném prostoru Wherigo. Zde je umístěn jen stručný seznam funkcí, objektů a konstant. Jejich detailní popis je umístěn na přiloženém CD.

Objekty

Bearing(value) Objekt reprezentující azimut

Distance(value, units) Objekt reprezentující vzdálenost v metrech, obsahuje možnost převodu na jiné jednotky

ZCommand(table) Akce objektu

ZReciprocalCommand Obousměrná akce objektu, použitá pro ukládané hry, málo využívané

ZonePoint(lat, lon, alt) Bod se zadanými geografickými souřadnicemi a nadmořskou výškou

ZObject Bázový objekt pro následující objekty

Zone Zóna složená z alespoň tří bodů ZonePoint

ZCartridge Objekt cartridge. Měl by existovat jediný v celém prostředí

ZMedia Objekt reprezentující média uložená v cartridge jako další objekty

ZItem Objekty, které může hráč držet nebo vidět

ZTask Objekt úkolu

ZTimer Objekt časovače – vyžaduje interakci s C++

ZInput Objekt pro vstup uživatele

ZCharacter Objekt postavy nebo hráče

Funkce

MessageBox(table) Zobrazení zprávy uživateli

Dialog(table) Zobrazení série zpráv uživateli

PlayAudio(media) Požadavek na spuštění přehrávání audia

ShowStatusText(text) Zobrazení stavové zprávy. Nevyužívané

VectorToZone(point, zone) Počítá vzdálenost z bodu k nejbližšímu bodu zóny

VectorToSegment(point, p1, p2) Počítá vzdálenost z bodu `point` k segmentu zóny. Využito funkcí výše

IsPointInZone(point, zone) Určuje, zda se bod vyskytuje v zóně

TranslatePoint(point, distance, bearing) Posun bodu o danou vzdálenost v daném azimutu

VectorToPoint(p1, p2) Výpočet vzdálenosti a azimutu mezi dvěma body

NoCaseEquals(s1,s2) Porovnání řetězců bez ohledu na velikost písmen

Command(command) Provedení příkazu – možnosti jsou SaveClose, DriveTo, StopSound, Alert

LogMessage(table) Záznam akce do logu cartridge

GetInput(input) Požadavek na vstup uživatele

ShowScreen(screen, item) Požadavek na zobrazení obrazovky

Konstanty

Konstanty pro funkci `ShowScreen`:

- MAINSCREEN
- INVENTORYSCREEN
- ITEMSCREEN
- LOCATIONSCREEN
- TASKSCREEN
- DETAILSCREEN

Konstanty úrovní logování pro funkci `LogMessage`:

- LOGDEBUG
- LOGCARTRIDGE
- LOGINFO
- LOGWARNING
- LOGERROR

Ostatní:

INVALID_ZONEPOINT Označení nezadaného objektu `ZonePoint`

Globální objekty:

Zdroj je wiki u projektu WherigoBuilder [22].

cartridge Objekt ZCartridge, který obsahuje aktuálně spuštěnou hru

Player Objekt ZCharacter, který představuje hráče hrajícího hru

Wherigo Výše popisovaná knihovna

Env Proměnné prostředí

WIGInternal Knihovna nativních funkcí, nevyužívaná samotnými cartridgemi