Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Diplomová práce

Multiplatformní konverze hry pro platformy iOS a Android

Plzeň 2013 Josef Vlach

Poděkování

Chtěl bych poděkovat své mamině za podporu a péči, kterou mi během celého studia věnovala.

Prohlášení

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne 13. srpna 2013

Josef Vlach

Abstract

Práce se zabývá konverzí počítačové hry Berušky na mobilní platformy iOS a Android. Probírána je obecná možnost psaní multiplatformního kódu pomocí dostupných knihoven a frameworků. Jsou zde popsány možnosti běhu C/C++ kódu na obou platformách. V realizační části práce je probrána možnost sdílení zdrojového kódu pro obě platformy a jsou zde probrány problémy spojené s implementací hry pro mobilní zařízení.

Obsah

1 Úvod

Cílem této práce je portace logické hry Berušky, napsané původně pro operační systém MS-DOS 5.x, na moderní mobilní operační systémy. Konkrétně se jedná a portaci na mobilní operační systém iOS od společnosti Apple Inc. a na mobilní operační systém Android od společnosti Google Inc. Cílem je udělat z původní DOSové hry moderní multiplatformní hru a přiblížit ji širokému publiku v podobě odpovídající moderním trendům. Práce byla zadána společností Red Hat, Inc. ve spolupráci s původním autorem hry.

Na začátku práce si podrobně představíme hru Berušky a poté si krátce popíšeme oba dva mobilní operační systémy, pro které portaci hry Berušky provedeme. Poté se v sekci Rešerše multiplatformních frameworků podíváme na to, jaké technologie se pro psaní multiplatformních her nabízejí. Projdeme rozlišné technologie, od HTML přes Flash až po jazyk C/C++. Po této sekci bude následovat podrobný popis obou dvou platforem a možností, které pro běh programů napsaných v jazyce C či C++ nabízejí. V realizační části práce se podrobněji podíváme na vývojová prostředí pro jednotlivé platformy a popíšeme si, jaké problémy multiplatformní vývoj přináší a jak je vyřešit. Závěrečné kapitoly budou patřit popisu změn ve hře Berušky, které byly v rámci portace z klasického počítače na mobilní telefon nutné učinit. Na závěr zhodnotíme dosažené výsledky této práce.

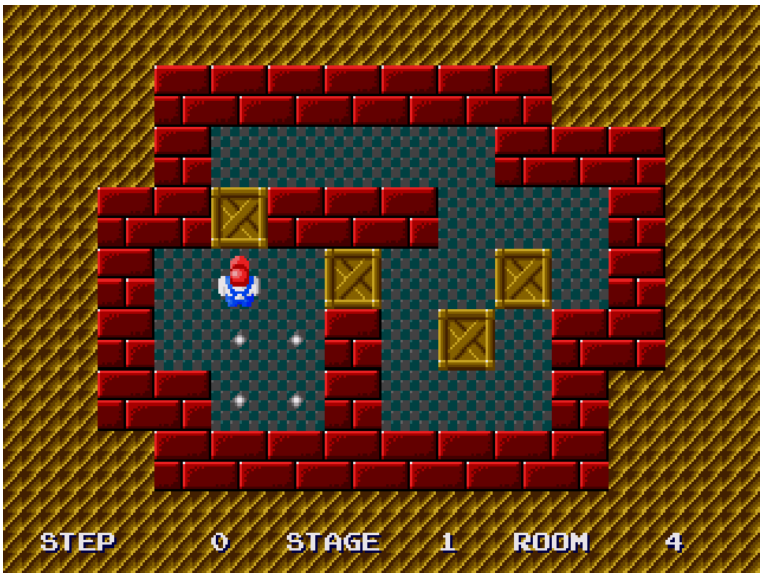
2 Teoretická část

2.1 Představení hry Berušky

Počítačová hra Berušky [16] je volně šiřitelná logická hra, vydaná pod GNU Public licencí. Hra Berušky, ukázka viz obrázek 2.1, vychází svými principy z logické herní klasiky Sokoban. Principem hry Sokoban je přemístit v herním levelu krabice z jejich počátečního rozložení do určených úložných míst. Hra Sokoban byla vytvořena japonským autorem jménem Hiroyuki Imabayashi v roce 1981 a následujícího roku došlo k jejímu komerčnímu vydání. Ukázka ze hry Sokoban viz obrázek 2.2. Ukázka je z verze hry z roku 1989 určené pro herní konzole Sega Mega Drive/Sega Genesis.

Obrázek 2.1: Ukázka ze hry Berušky

Hra je hrána na herním poli složeném ze čtverců, z nichž každý čtverec je podlaha, nebo stěna. Některé ze čtverců podlahy jsou označeny jako úložiště a některé obsahují krabice.



Obrázek 2.2: Ukázka ze hry Sokoban

Hráč se může pohybovat pouze po jednotlivých čtvercích herního pole horizontálně či vertikálně a to pouze na prázdná místa podlahy (nikdy skrze stěnu či krabici). Hráč může též odtlačit krabici na prázdný čtverec podlahy za krabicí. Krabice nemůže být tlačena do další krabice či stěny a též nemůže být tažena. Herní úroveň je vyřešena, když jsou všechny krabice uloženy v úložištích.

Sokoban byl studován v teorii složitosti a bylo dokázáno, že řešení Sokobanových úrovní je NP složitý problém. Další studie dokonce ukázala, že se jedná až o PSPACE-úplný problém [24].

Hra Berušky rozšiřuje herní princip Sokobanu o další herní mechanizmy a prvky, které si nyní uvedeme:

Výbušnina zničí krabici, na kterou je natlačena,

Kámen nelze odtlačit, ale lze ho zničit krumpáčem,

Krumpáč lze sebrat a následně použít na rozbití kamene,

Barevný klíč potřebný pro otevření barevné brány. Pouze beruška shodné barvy jej může sebrat,

Barevné dveře lze je otevřít pouze za použití odpovídajícího barevného klíče,

Barevná brána brána, kterou může projít pouze beruška s odpovídající barvou,

Brána pro jedno použití brána, kterou lze projít pouze jednou.

Vedle těchto prvků je hlavní designovou změnou hry Berušky rozšíření hry o kooperativní prvek, při kterém hráč ovládá až pět rozlišných berušek (beruška je hratelná postava) a jednotlivé berušky si musejí vzájemně pomáhat k úspěšnému vyřešení jednotlivých herních úrovní.

Další konceptuální odlišnost od původního Sokobanu je nutnost sebrání pěti klíčů, které odemykají dveře od domečku, ke kterému se musí alespoň jedna z berušek dostat, což samo o sobě může být veliká výzva. Není tudíž nutno umisťovat jednotlivé krabice do úložných míst.

Hra byla vyvinuta českým nezávislým herním týmem Anakreon [7] pod vedením Martina Stránského. Hra byla původně vydána pro MS DOS 5.x, ale pro svůj úspěch byla později předělána na další operační systémy a nyní Berušky běží na operačním systému Linux i Windows. Pro grafické pozadí je využito multimediální knihovny SDL. Hra obsahuje 120 herních úrovní.

2.2 Představení mobilních platforem

Pro naplnění předmětu této práci si nyní představíme obě dvě mobilní platformy, na které hru Berušky převedeme. Povíme si něco o základních rysech a historii.

2.2.1 iOS

Operační systém iOS dříve iPhone OS, viz obr. 2.3, je operační systém určený pro mobilní zařízení, který byl a neustále je vyvíjený a distribuovaný společností Apple Inc. Jeho prvotní představení se odehrálo v roce 2007 společně s mobilním telefonem iPhone. Postupně byl systém iOS rozšířen na ostatní Apple zařízení, jako iPod Touch (září 2007), iPad (leden 2010) a druhou generaci Apple TV (září 2010). V průběhu těchto let se zároveň s uváděním

dalších zařízení, na kterých iOS běžel, či nových verzí stávajících zařízení, vyvíjela i verze samotného systému iOS. První verzí byla verze 1.0, která se objevila současně s uvedením prvního mobilního telefonu iPhone. Tato verze prošla postupně několika updaty, z nichž poslední nesl označení 1.1.5. Postupně v průběhu let následovali verze 2.x, 3.x, 4.x, 5.x a 6.x. V době psaní této práce je aktuální verze operačního systému iOS verze s označením 6.1.3. V červnu 2013 byla oznámena verze iOS 7.0, která se objeví na podzim roku 2013 [44].

Obrázek 2.3: Ukázka operačního systému iOS

Na rozdíl od systému Windows Phone od společnosti Microsoft či Android od společnosti Google nenabízí Apple systém iOS ostatním výrobcům hardwaru a jedná se tedy o uzavřenou platformu, která se i přesto těší ve světě velké oblibě.

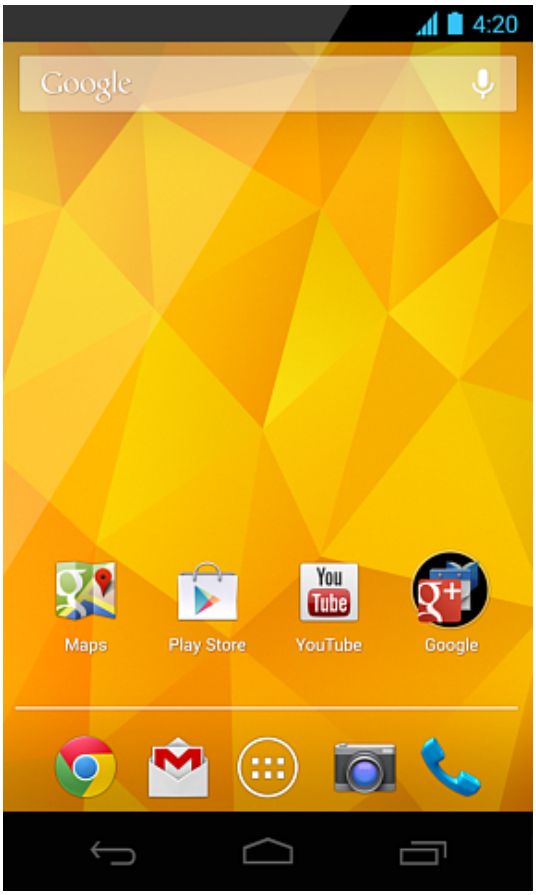
Uživatelské rozhraní operačního systému iOS je založeno na konceptu přímé manipulace za použití multidotykových gest. Prvky uživatelského rozhraní tvoří posuvníky, přepínače a tlačítka. Interakce s operačním systémem probíhá pomocí gest jako *swipe* (táhnutí prstem), *tap* (poklepání prstem), *pinch* (přiblížení dvou prstů), či *reverse pinch* (oddálení dvou prstů), které mají speciální význam v závislosti na kontextu operačního systému a jeho multidotykového rozhraní.

iOS je mobilní verzí operačního systému OS X známého z osobních počítačů Apple. Z důvodu uzavřenosti platformy a díky tomu, že společnost Apple vyvíjí pro své telefony (a ostatní zařízení, které jsou poháněny systémem iOS) i vlastní hardware, dokáže společnost Apple maximálně vyladit a optimalizovat spolupráci mezi tímto hardwarem a softwarem a tím nabídnout uživatelům nejvíce optimalizovaný operační systém na trhu. K tomu napomáhá i vlastnictví několika důležitých patentů jako například známý patent číslo 7,786,975 [8]. Pomocí tohoto patentu je docíleno plynulého ovládání tažením bodem kontaktu, zejména v situacích, kdy se dostaneme na konec obsahu na obrazovce a dále již není co zobrazovat. V ten moment je možné s obsahem dále táhnout, ale pouze do určité úrovně a čím dále táhneme, tím máme vizuální pocit stále většího tření a tažení dále je stále náročnější a po puštění bodu tažení se obsah plynule zhoupne zpět, tak aby zaplnil celou plochu telefonu, jež má k dispozici.

2.2.2 Android

Android (viz obr. 2.4) je operační systém od společnosti Google Inc., založený na operačním systému Linux. Android je primárně zaměřen na mobilní zařízení vybavené dotykovou obrazovkou jako například chytré telefony nebo tablety. Android byl původně vyvinut společností Android Inc., jejímž hlavním investorem byla společnost Google. Později v roce 2005 došlo ke koupi společnosti Android Inc. právě společností Google. Operační systém Android byl představen v roce 2007 a zároveň s jeho představením byla založena aliance Open Handset Alliance [6]. Jedná se o konsorcium hardwarových, softwarových a telekomunikačních společností věnující se prosazování otevřených standardů v oblasti mobilních zařízení.

V době svého prvního uvedení se nacházel systém Android v beta verzi. První komerční verze se objevila v září 2008 a jednalo se o verzi 1.0. Od té doby doznal systém Android řady větších či menších vylepšení od změny



Obrázek 2.4: Ukázka operačního systému Android

linuxového jádra z verze 2.6.x na verzi 3.0.x, přes různá zrychlení systému, paměťové optimalizace, až po kompletní přepracování uživatelského ovládání. V červenci 2013 byla představena verze Androidu 4.3 s kódovým označením Jelly Bean.

Operační systém Android je open source a je vydán pod Apache licencí. To umožnuje, aby byl zdrojový kód volně upravován a distribuován výrobci zařízení a zároveň v případě proprietárních vylepšení jednotlivých výrobců, nemuseli výrobci tato vylepšení vydávat zpátky open source komunitě a tudíž si každý výrobce může chránit svoje know-how.

V době svého představení nebyl Android vnímán jako vážná hrozba [11], v té době zaběhnutými společnostmi na mobilním trhu jako Nokia, Microsoft či Apple. Úspěch Androidu ale všechny překvapil, což na jedné straně vedlo

k ústupu ze slávy do té doby dominantní Nokie a na druhé straně ke spoustě patentových sporů okolo Androidu, z nichž nejznámější je spor mezi Googlem a Oraclem ohledně porušení autorských práv Oraclu týkajících se využití jazyka Java pro běh uživatelských aplikací [41].

2.3 Rešerše multiplatformních frameworků

V případě vývoje pro mobilní platformy je nutné brát v potaz, proč zamýšlenou aplikaci děláme a co je naším cílem. Pokud je naším cílem oslovit pouze uživatele jedné konkrétní platformy, nemusíme se starat, jestli námi zvolený *framework*, tj. sada knihoven a nástrojů usnadňujících vývoj pro danou platformu, podporuje možnost výslednou aplikaci převést i na jiné platformy. V opačném případě, kdy zamýšlíme výslednou aplikací oslovit uživatele více mobilních platforem, je dobré zvolit takový framework, který nám vývoj pro více platforem dovolí. Takový framework nazýváme *multiplatformní framework*. Kdybychom takový framework nepoužili a chtěli bychom napsat multiplatformní aplikaci, znamenalo by to napsat tuto aplikaci pro každou platformu zvlášť, což se s rostoucím počtem platforem stává neúnosným a v této situaci je vidět výhoda multiplatformních frameworků.

Při výběru vhodného multiplatformního herního frameworku nabízí trh širokou škálu možností. Dle klíčových technologií, které tyto frameworky využívají je lze rozdělit do tří základních skupin: HTML5/CSS, Adobe AIR (Flash) a 2D/3D frameworky. V následující části si přiblížíme zástupce z každé této skupiny.

2.3.1 HTML5/CSS herní frameworky

První potencionální skupinu technologií pro multiplatformní vývoj tvoří webové technologie, kdy se využívá běhu aplikací ve webovém prohlížeči. Během ve webovém prohlížeči není myšleno, že by aplikace běžela na nějakém webovém serveru a přistupovalo se k ní přes www stránky pomocí webového prohlížeče, jako například pomocí prohlížeče Chrome od Googlu, či Safari od Applu. Ačkoliv toto je taky jeden z možných přístupů, my máme na mysli aplikace, které pro svůj běh využívají webový prohlížeč na svém pozadí a jejich instalace probíhá klasickým způsobem přes obchody s aplikacemi jednotlivých platforem (App Store v případě iOS a Google Play v případě Androidu).

Sencha Touch

Sencha Touch [43] je javascriptová UI knihovna navržená speciálně pro psaní mobilních aplikací. Jejím typickým použitím je psaní mobilních webových aplikací, které mají *look and feel* jako nativní aplikace na podporovaných zařízeních. Je zcela založená na webových standardech jako HTML5, CSS3 a JavaScript. Zaměřením Sencha Touch knihovny je nabídnutí vývojářům rychlé a snadné vytvoření HTML5 aplikací, které fungují na Androidech, iOS a BlackBerry zařízeních a poskytnout zkušenost známou z nativních aplikací uprostřed webového prohlížeče.

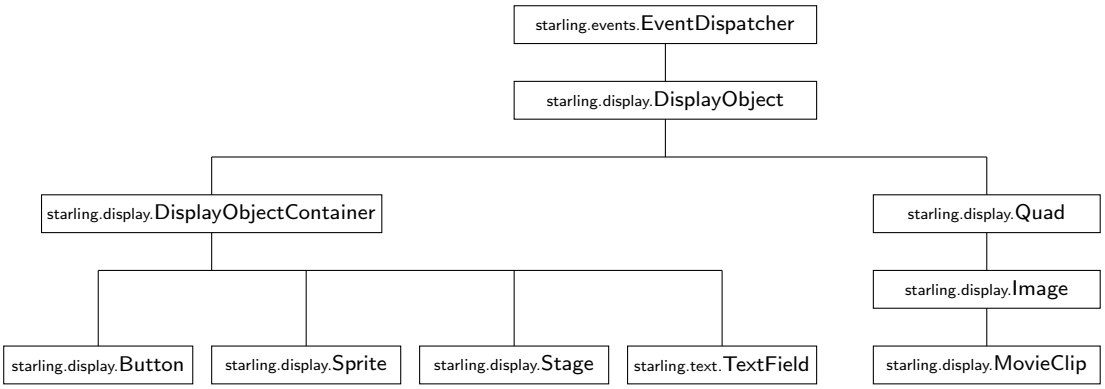
Apache Cordova

Apache Cordova [15] je open source mobilní vývojový framework umožňující využití standardních webových technologií jako HTML5, CSS3 a JavaScript pro multiplatformní vývoj. Umožnuje tak vyhnout se nativnímu jazyku každé mobilní platformy. Zároveň s tím ale nabízí možnost přístupu k sensorům mobilních zařízení, jako například akcelerometr, kamera, kompas atd. pomocí javascriptových API. Je možná kombinace z různými UI frameworky jako například jQuery Mobile [25], Dojo Mobile [29] nebo Sencha Touch [43].

Apache Corodova je k dispozici pro tyto platformy: iOS, Android, Blackberry, Windows Phone, Palm WebOS, Bada a Symbian.

2.3.2 Adobe AIR (Flash) herní frameworky

Další možným řešením multiplatformního vývoje je technologie Adobe Integrated Runtime (Adobe AIR) [5] od společnosti Adobe Systems Inc. Adobe AIR je běhové prostředí, pro které lze vyvíjet aplikace za pomocí Adobe Flash, Apache Flex (dříve Adobe Flex), HTML a Ajax. Aplikace napsaná pro Adobe AIR může být nasazená jak na desktopové prostředí, tak na mobilní zařízení. Z desktopových prostředí je podporováno Windows (XP a novější) a OS X. Oficiální podpora pro Linux byla ukončena v červnu roku 2011. Mezi podporované mobilní operační systému patří Android (od verze 2.3), Apple iOS (od verze 4.3) a BlackBerry Tablet OS. Programovacím jazykem prostředí Adobe AIR je ActionScript, což je dialekt ECMAScriptu.



Obrázek 2.5: Hierarchie tříd zobrazovacích objektů

Starling

Starling [42] je open source herní engine využívající Adobe AIR. Jak bylo řečeno v předchozím odstavci, Adobe AIR využívá Adobe Flash a fundamentální součástí Adobe Flashe je tzv. display tree architecture viz obrázek 2.5. Ta popisuje jak a kam kreslit objekty na obrazovku. Starling byl inspirován touto architekturou do té úrovně, že mnoho tříd a funkcí mají stejná jména jako jejich originální Flashové ekvivalenty.

Kořenová třída všeho co lze zobrazit na obrazovku je `DisplayObject`. Jedná se o abstraktní třídu, takže od ní nelze vytvořit instanci. Ale poskytuje metody a vlastnosti, které každý zobrazovací objekt sdílí, viz následující výčet:

- position (x, y)
- size (width, height)
- scale factor (scaleX, scaleY)
- rotation
- opacity (alpha)

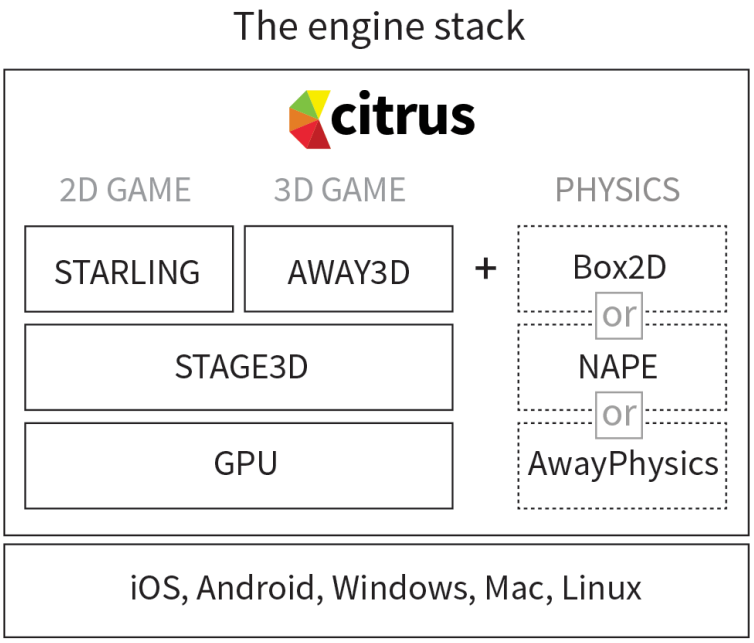
Podíváme-li se blíže na konkrétní třídy dědící z `DisplayObject`, vidíme několik tříd, jež můžou být použity, tak jak jsou: `Quad`, `Image`, `TextField`, `Button` či `MovieClip`. Je poměrně zřejmé co s každou třídou můžeme docílit.

Další abstraktní třídou je `DisplayObjectContainer`, která slouží jako kontejner pro další objekty, které chceme zobrazit. Dovoluje organizovat zobrazované objekty do logického systému - tzv. display tree, kde pomocí jednodušších objektů můžeme skládat objekty složitější.

Programovacím jazykem knihovny Starling je ActionScript 3.

Citrus

Citrus [19] je multiplatformní herní engine akcelerovaný pomocí GPU což zajištuje Stage3D [3] technologie od firmy Adobe. Pro zobrazení 2D grafiky se využívá knihovna Starling (představená v předešlé sekci). Pro zobrazení 3D grafiky je využívá engine Away3d [18].



Obrázek 2.6: Architektura Citrus Enginu [19]

Vedle 2D či 3D grafiky nabízí tento engine i podporu fyziky, kde si lze vybrat z několika fyzikálních knihoven: Box2D [13], Nape [20], AwayPhysics [33] a jednoduchý detektor kolizí.

Dalšími z podporovaných nástrojů pro jednodušší vývoj her jsou editory levelů: Tiled Map Editor [17] a Generic LEvel EDitor 2D [2].

2.3.3 C/C++ herní frameworky

Dalším a pravděpodobně nejrozšířenějším řešením multiplatformního vývoje pro mobilní systémy je využití C/C++ frameworků. Tyto frameworky většinou nabízejí možnost vývoje 2D a 3D grafiky pomocí OpenGL ES rozhraní.

OpenGL ES

OpenGL for Embedded Systems (OpenGL ES) [21] je podmnožinou OpenGL 3D grafického aplikačního programového rozhraní navrženého pro vestavěné systémy jako například mobilní telefony, PDA zařízení či kapesní herní konzole (např. Nintendo 3DS).

V současné době existuje několik verzí OpenGL specifikace. OpenGL ES 1.0 je vztaženo k OpenGL 1.3, OpenGL ES 1.1 je odvozeno od OpenGL 1.5 specifikace a OpenGL ES 2.0 je odvozeno od OpenGL 2.0 specifikace. To znamená, že například aplikace napsaná pro OpenGL ES 1.0 by měla být jednoduše portovatelná do desktopového OpenGL 1.3, jelikož OpenGL ES je zredukovaná verze tohoto API. Opak může a nemusí být pravda, záleží na použití jednotlivých vlastností.

Cocos2D Python

Cocos2D je 2D open source herní framework. Původní Cocos2D framework je napsán v Pythonu, ale byl portován do dalších jazyků a na další platformy. Cocos2D (Python) umožnuje psát hry pro platformy Mac, Windows a Linux a na těchto třech platformách lze také hry vyvíjet. Svou jednoduchostí použití, tím jak jednoduše lze vytvářen animace, přechody, jednoduchou prací se sprity, podporující operace jako *move*, *rotate*, *scale*, *sequence*, *spawn* atd., si tento framework získal velkou popularitu a dočkal se mnoha portů na jiné platformy.

Cocos2D iPhone

Vůbec nejznámějším a nejúspěšnějším portem je Cocos2d iPhone. Název sám již napovídá, že jde o port na platformu iOS a kromě této platformy lze pomocí tohoto portu hry publikovat ještě pro Mac. Cocos2D iPhone se dočkal

takové popularity, že se dnes v podstatě stal synonymem pro pojem Cocos2D. Pokud se tedy někde mluví o Cocos2D, mají lidi většinou na mysli Cocos2D iPhone, nikoliv původní pythonovský Cocos2D.

Rozdíl mezi pythonovským Cocos2D a Cocos2D iPhone je zejména v tom, že Cocos2D iPhone je kompletně napsaný v programovacím jazyce Objective-C a lze v něm vyvíjet pouze v prostředí operačního systému Mac OS. Cocos2D iPhone má kolem sebe jak rozsáhlou komunitu uživatelů, tak i vývojářů a bylo v něm naprogramováno mnoho úspěšných titulů jako například úspěšná hra Badland [10].

Cocos2D je herní API, které využívá OpenGL ES, je snadno rozšiřovatelné a jednoduše integrovatelné s knihovnami třetích stran. Nabízí též snadnou integraci s fyzikálními knihovnami Chipmunk či Box2D.

Při řešení našeho problému je nezbytná vedle iOS též podpora platformy Android. Již z názvu tohoto portu Cocos2D frameworku je patrné, že to s Platformou Android nebude vůbec jednoduché. A to se také ukazuje v praxi. Cocos2D portování na Android nepodporuje. Existuje sice služba Apportable [9], pomocí které lze aplikace napsané v Cocos2D iPhone převést na platformu Android, ale jedná se o komerční službu a ta jako taková není zadarmo. Naštěstí popularita Cocos2D iPhone vzrostla natolik, že se objevili další porty Cocos2D založené právě na Cocos2D iPhone.

Cocos2D-x

Cocos2D-x je open source multiplatformní 2D herní framework, který vychází z Cocos2D iPhone. Podporované platformy tohoto frameworku jsou:

iOS stabilní, otestovaný na iOS 5.x $\sim$ 6.x SDK.

Android stabilní, otestovaný na 2.0 $\sim$ 4.x, ndk r5 $\sim$ r8

Windows Phone 8 a Win8 Metro stabilní

Bada dále nepodporováno od cocos2d-x v2.0

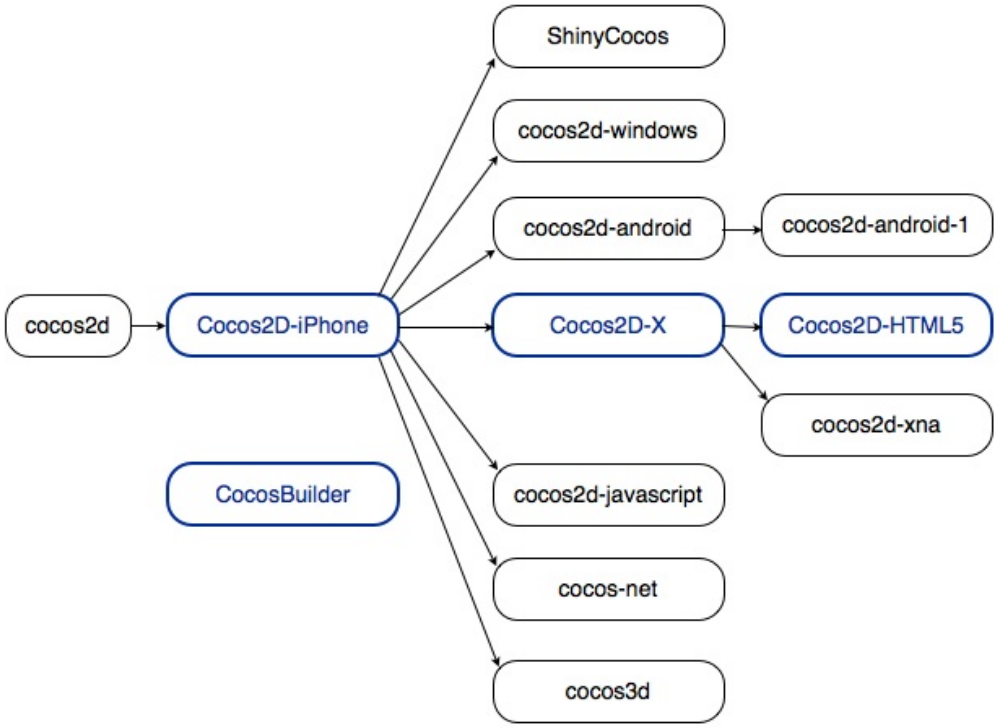
BlackBerry stabilní, podporuje Playbook a BB10

Marmalade stabilní od cocos2d-x v0.11.0

Windows stabilní, otestováno na WinXP/Vista/Win7

Linux podporován, ale občas nestabilní

Hlavním programovacím jazykem Cocos2D-x je jazyk C++. Vedle jazyka C++ ale nabízí též binding pro jazyky Lua a Javascript a díky tomu lze hry snadno psát i pro cílovou platformu HTML5. Na následujícím obrázku 2.7 je zobrazena rodina Cocos2D frameworků (pouze ty nejznámější větve).



Obrázek 2.7: Vztahy mezi Cocos2d frameworky

Frameworky zobrazené modře (Cocos2d-iPhone, Cocos2d-x, Cocos2d-HTML5, a Cocos Builder [14]) vzájemně spolupracují v tom smyslu, že nové verze těchto frameworků vycházejí koordinovaně s frekvencí cca 4 releasů do roka.

Marmalade

Marmalade SDK [39] je multiplatformní framework od firmy Ideaworks3D Limited. Marmalade SDK byl dříve znám pod názvem Airplay SDK, ale

v červenu 2011 společně s vydáním verze 5.0.0. došlo k přejmenování na Marmalade SDK. SDK bylo nejdříve využíváno a vyvíjeno pro interní vývoj video her ve společnosti Ideaworks3D, ale později se změnilo na samostatný produkt.

Základní myšlenkou Marmalade SDK je napiš jednou, spusť všude, tak aby jeden kód mohl být zkompilován a spuštěn na všech podporovaných platformách, místo toho, aby bylo nutné pro každou platformu psát program v jiném jazyce za použití rozlišných API. Tohoto je docíleno poskytnutím C/C++ API, které slouží jako abstraktní vrstva pro klíčové API jednotlivých platforem.

V únoru 2013 byl vydán RAD nástroj Marmalade Quick. Jedná se o nástroj pro rychlý a flexibilní vývoj 2D her a aplikací. Je založen na nejlepších open source komponentách jako například Cocos2D-x či fyzikálním enginu Box2D. Jako programovací jazyk je použit scriptovací jazyk Lua [28].

Pomocí Marmalade frameworku bylo naprogramováno spoustu úspěšných her, mezi nimiž je i jedna z nejúspěšnějších her na mobilní platformy hra Cut the Rope [37]. Nevýhodou tohoto frameworku je nutnost pořízení licence.

Podporované platformy jsou:

iOS Všechny licence

Android Všechny licence

BlackBerry PlayBook Indie licence a vyšší

LG Smart TV Profesionál licence a licencovaní LG vývojáři

Mac OS X Plus licence a vyšší

Windows Plus licence a vyšší

Windows Phone 8 Indie licence a vyšší

Corona SDK

Corona SDK [38] multiplatformní framework pro vývoj aplikací pro mobilní platformy iOS, Android, Amazon Kindle Fire a Barnes & Noble NOOK.

Corona SDK využívá programovací/skriptovací jazyk Lua jako vrstvu abstrakce nad C++/OpenGL částí knihovny. V dubnu 2013 byla uvolněna verze Corona SDK Free, pomocí které lze programovat a následně publikovat hry zcela zdarma. Zpoplatněny zůstaly speciální funkce jako přístup k InApp nákupům apod.

Corona SDK podporuje rozlišné nástroje třetích stran pro ještě rychlejší a komfortnější vývoj aplikací. Mezi těmito nástroji jsou například:

Lua Glider je IDE pro jazyk Lua obsahující vše co programátor od moderního IDE očekává včetně debuggeru.

Corona Complete je vizuální debugger, editor kódu a projektový manažer. Nabízí chytrý autocomplete, konzolový výstup, management souborů, error a breakpoint management.

SpriteHelper je aplikace pro MAC OS pomáhající s tvorbou textur, animací a fyzikálního modelu.

LevelHelper je editor úrovní pro MAC OS, který lze snadno používat dohromady se SpriteHelper.

Particle Candy je částicový editor pro tvorbu speciálních efektů.

PhysicsEditor usnadňuje vývojáři vytvářet a editovat fyzikální tvary herních entit.

V základní neplacené verzi nabízí Corona SDK vynikající možnosti pro vývoj na mobilní zařízení a obzvláště pokud chceme program cílit nejenom na nejrozšířenější platformy iOS a Android, ale také na stále populárnější čtečky knih, je Corona SDK ideální volbou.

Unity

Unity (též známé pod názvem Unity3D) je multiplatformní herní engine obsahující své vlastní IDE, viz obrázek 2.8, vyvinuté společností Unity Technologies. Unity lze využít jak pro vývoj webových, desktopových a konzolových her, tak pro vývoj her na mobilní platformy. Poslední verze Unity 4.2, která byla vydána v červenci 2013, podporuje konkrétně tyto platformy: iOS, Android, Windows, Blackberry 10, MAC OS X, Linux, webové prohlížeče, Flash, PlayStation 3, Xbox 360, Windows Phone 8 a Wii U.

První verze Unity byla představena v roce 2005 na Apple Worldwide Developers Conference (WWDC). V té době engine Unity podporoval pouze platformu MAC OS X, ale dočkal se velké podpory od vývojářů, takže se postupně rozšířil i na další platformy. Unity 3 bylo představené v roce 2010 se zaměřením na poskytnutí vývojářům nástroje odpovídající tomu, co mají k dispozici velké herní studia. Tím si získalo ještě větší pozornost a nejen od malých studií, ale také od velkých vývojářských domů.

Herní engine Unity je k dispozici ke stažení ve dvou rozlišných verzích: Unity a Unity Pro.

Jakou verzi Unity si pořídit závisí na tom, na kterou platformu chtějí vývojáři svojí hru cílit. Platformy jako PlayStation 3, Xbox 360 jsou k dispozici pouze s Unity Pro a do nedávné doby byla tato verze jediná možnost jak vyvíjet pro platformy iOS a Android. Dne 21. května 2013 ale přišla změna a od této chvíle je základní verze Unity k dispozici zdarma i pro mobilní platformy iOS a Android [46].

Unity využívá pro svůj grafický engine Direct3D (Windows, XBOX 360), OpenGL (Mac, Windows, Linux, PS3), OpenGL ES (Android, iOS), a proprietární API (Wii).

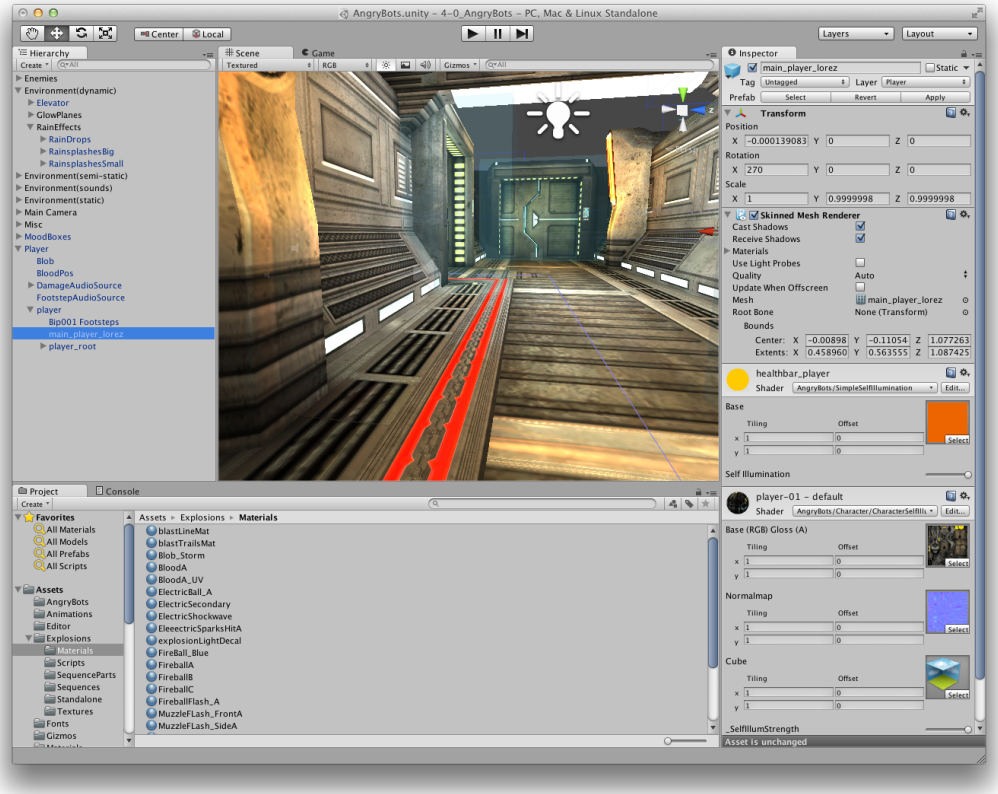
Unity podporuje souborové formáty modelů, textur a jiných herních prvků z následujících grafických programů a nástrojů: 3ds Max, Maya, Softimage, Blender, modo, ZBrush, Cinema 4D, Cheetah3D, Adobe Photoshop, Adobe Fireworks a Allegorithmic Substance. Tyto herní modely, textury atd. mohou být přidány do projektu a spravovány pomocí grafického uživatelského rozhraní systému Unity.

Unity podporuje nativně tři programovací jazyky:

C# je univerzální objektově orientovaný programovací jazyk vyvinutý společnosti Microsoft.

UnityScript jazyk speciálně navržený pro použití s Unity. Jedná se o jazyk po syntaktické stránce podobný Javascriptu, ale sémanticky se jedná o zcela odlišný jazyk. (V Unity komunitě jsou pojmy UnityScript a Javascript synonyma).

Boo objektově orientovaný, staticky typovaný jazyk z rodiny .NET se syntaxí inspirovanou jazykem Python.



Obrázek 2.8: Ukázka Unity IDE

Unity též nabízí podporu fyzikálního enginu PhysX od společnosti Nvidia s podporou simulace šatů v reálném čase.

Přesto, že je Unity 3D herní framework, lze pomocí něj vytvářet i 2D hry. Jedním z nejúspěšnějších příkladů je hra Bad Piggies [34] od známého herního studia Rovio.

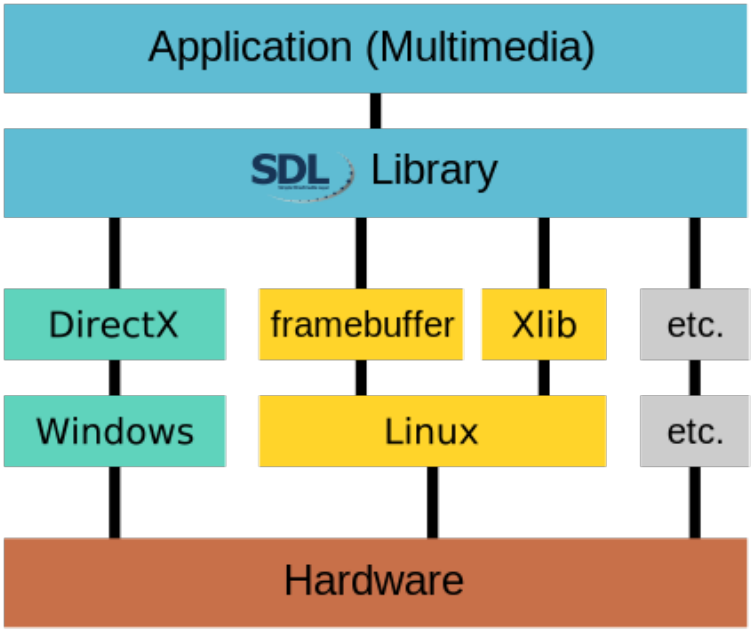
SDL

SDL neboli Simple DirectMedia Layer [27] je open source multiplatformní multimediální knihovna navržená tak, aby poskytla nízko úrovňový programový přístup k souborovému systému, vláknům, počítačové síti, k hardwarovým periferiím jako je klávesnice, myš, joystick, CD-ROM, audio atd., dále

přístup k 3D hardwaru prostřednictvím OpenGL API a v neposlední řadě přístup přímo k 2D video framebufferu.

SDL podporuje Mac OS X, Linux, Windows, FreeBSD, NetBSD, OpenBSD, BSD/OS, Solaris, iOS a Android. Zdrojový kód obsahuje též podporu pro další operační systémy (např. AmigaOS, Dreamcast, Atari, AIX, OSF/Tru64, RISC OS, SymbianOS a OS/2), ale tyto nejsou oficiálně podporovány a z dnešního pohledu už se jedná pouze o raritu určenou milovníkům zašlých časů.

SDL je napsáno v jazyce C, ale pracuje nativně s jazykem C++ a obsahuje binding pro spousty dalších jazyků, např. Ada, C#, Eiffel, Erlang, Euphoria, Go, Guile, Haskell, Java, Lisp, Lua, ML, Objective-C, Pascal, Perl, PHP, Pike, Pliant, Python, Ruby a Smalltalk.



Obrázek 2.9: Architektura Simple DirectMedia Layer

SDL má ve svém názvu slovo "layer" (vrstva), protože z technického hlediska se jedná o wrapper okolo specifických funkčností operačních systémů viz obrázek 2.9. Hlavním účelem SDL knihovny je poskytnout společný přístup k těmto funkcionalitám.

2.3.4 Ostatní frameworky

Vedle právě probraných možností pro multiplatformní mobilní vývoj herních aplikací existuje ještě celá řada dalších, zde neprobraných možností, z nichž se na jedné straně jedná povětšinou o placené frameworky, čímž se jejich použití pro účely této práce stává nepraktické. Na druhou stranu se jedná o skupinu frameworků a knihoven které jsou sice zdarma, ale u kterých by jejich osvojení a naučení bylo velice obtížné z důvodu jejich menšího rozšíření, nedostatečné dokumentace a dalších důvodů postihující malé projekty. U spousty z těchto ostatních knihoven hrozí též nebezpečí, že se dále nebudou rozšiřovat, neboť se povětšinou jedná o projekty, za kterými stojí malý počet lidí (one man show projekty nejsou výjimkou).

2.3.5 Vyhodnocení rešerše

Ukázali jsme, že nabídka frameworků pro multiplatformní vývoj pro mobilní zařízení je široká a když se zaměříme pouze na dvě hlavní platformy iOS a Android, je nabídka ještě širší, což je způsobeno hlavně procentuálním zastoupením obou platforem na mobilním trhu, kde obě dvě platformy dohromady celému trh jasně dominují [30].

Zodpovědět otázku, kterou technologii pro multiplatformní vývoj her zvolit není jednoduché. Je zapotřebí zvážit povahu hry, kterou plánujeme na mobilní platformy přinést, zdali se jedná o 3D hru nebo si vystačíme s jednodušším 2D. Další otázkou je, zdali bude potřeba podpory fyzikálního enginu, či charakter hry s takovou funkcionalitou nepočítá. Neméně důležitou otázkou před začátkem projektu je zvážení případného budoucího rozvoje i na jiné platformy nežli iOS a Android. A v neposlední řadě je zapotřebí vzít do úvahy schopnosti a možnosti všech členů týmu.

Další alternativou je začít na zelené louce a napsat vlastní framework, jež bude pokrývat všechny potřeby našeho projektu. Pro tuto alternativu je ale rozumné se rozhodnou jen po předchozích zkušenostech a se znalostí omezení stávajících řešení. Z ekonomického hlediska je toto řešení nejméně výhodné a je velice nepravděpodobné, že by stávající frameworky nepokryli potřeby našeho projektu alespoň do určité míry. V našem případě je ale tato možnost první, kterou musíme zavrhnout z důvodu nedostatečných znalostí probraných frameworků, z nichž některé nabízí skutečně lákavé vlastnosti.

V našem případě máme dané koncové platformy iOS a Android, takže můžeme vybrat jakýkoliv z představených frameworků. Co se týče fyziky, tak žádné fyzikální prvky hra Berušky neobsahuje. Stále jsme, tudíž v situaci kdy můžeme zvolit libovolný z frameworků. V našem případě je ale velice důležitým faktorem to, že se jedná o konverzi hry, jež je napsaná v C++ pomocí knihovny SDL. Verze knihovny SDL ve které jsou Berušky naprogramovány je 1.2. Kořeny této verze sahají až do roku 2000, tedy do doby kdy pojem chytrý telefon nikomu nic neříkal a tak není divu, ža ani v této verzi knihovny SDL jakákoliv podpora mobilních platforem schází. To všechno se změnilo s příchodem SDL verze 2.0, jež byla oznámena 14. července 2012 [22]. Framework SDL 2.0 (původní označení bylo SDL 1.3) je značné rozšíření původního kódu SDL 1.2, které není s touto verzí zpětně kompatibilní. Framework SDL 2.0 je vydán pod zlib licencí [47]. Z důvodu značné závislosti Berušek na tomto frameworku (zejména v oblasti renderování výsledné scény) by volba jiného frameworku znamenala kompletní přepsání celé hry Berušky, takže by výsledný kód s tím původním měl jen málo společného. Z tohoto důvodu jsme se rozhodli pro řešení povýšením verze frameworku SDL z verze 1.2 na verzi 2.0 a vyřešit problémy s nekompatibilním API, které se ve zdrojových kódech hry Berušky objeví. Ačkoliv očekáváme, že těchto nekompatibilit se objeví celá řada, stále jejich oprava bude vyžadovat nepatrné množství času oproti množství času, co by si vyžádalo kompletnímu přepsání zdrojových kódů. Takto ušetřený čas věnujeme předělání ovládání a uživatelského prostředí pro dotykové telefony.

3 Popis platforem Android a iOS

3.1 Popis architektury Androidu

Jak již bylo řečeno v kapitole 2.2.2, operační systém Android je založen na operačním systému Linux verze 2.6. S verzí Androidu 4.0 přišla změna i v této oblasti a Android v této verzi vychází z verze Linuxu 3.x spolu s middleware knihovnami viz obrázek 3.1.

Linuxové jádro slouží jako abstrakce nad hardwarem, pro správu paměti a další nízko úrovňové věci a společně s middleware knihovnami je tato část Androidu napsána v jazyce C. Aplikační software naproti tomu běží za pomocí aplikačního frameworku, který je napsán v jazyce Java a tím pádem i primární programovací jazyk pro aplikační software pro Android je jazyk Java. Aplikační framework poskytuje aplikacím služby jako například:

Activity Manager stará se o životní cyklus aplikace.

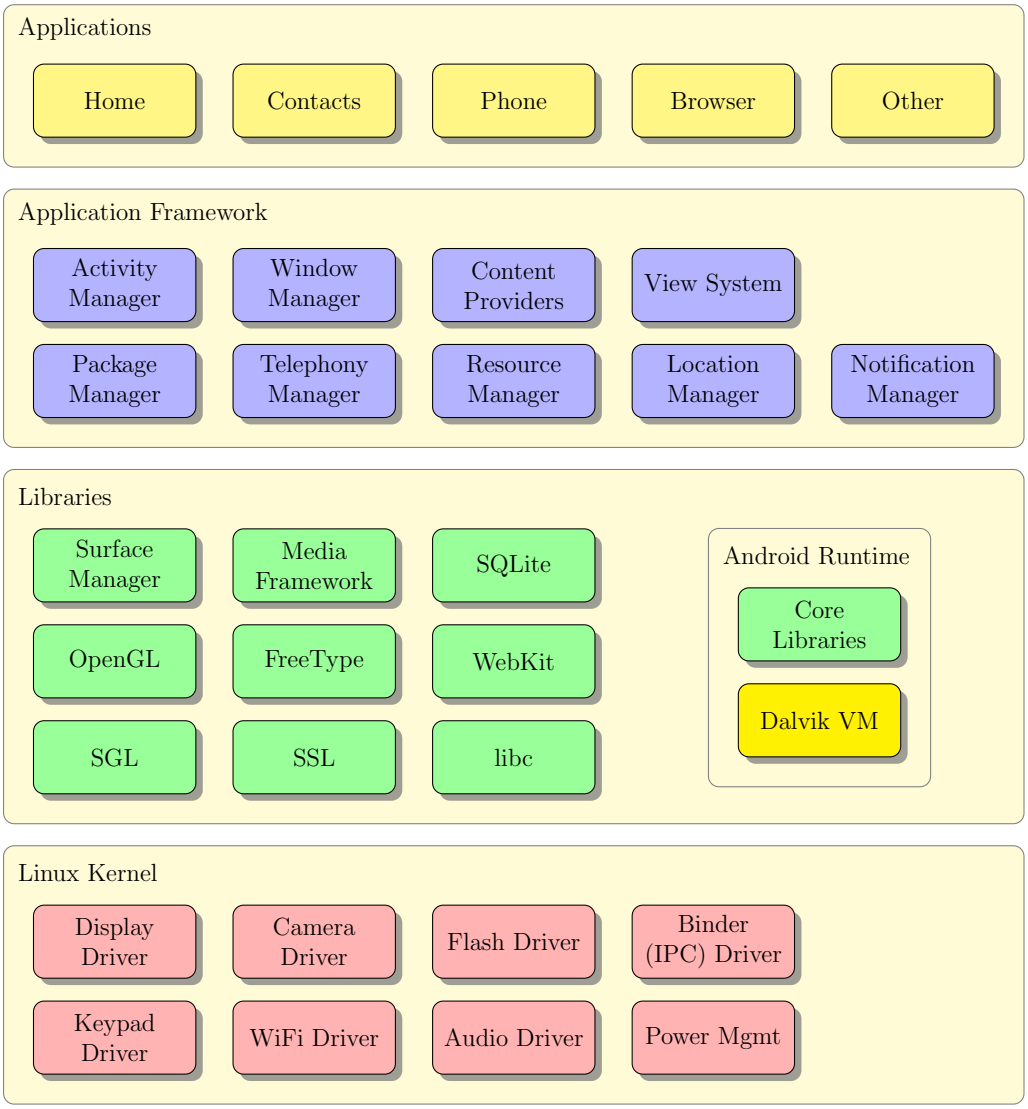
Content providers zapouzdřuje data, která jsou sdílená (např. seznam kontaktů).

Resource manager stará se o všechny zdroje.

Location manager poskytuje lokaci telefonu (pomocí GPS, GSM, WiFi).

Notification manager spravuje všechny události, jako např. příchod sms, schůzky atd.

Běh samotných Java aplikací je zajištěn pomocí virtuálního stroje Dalvik [45] obsahující just-in-time kompilátor. Na rozdíl od virtuálního stroje Javy, který vykonává program po jeho kompilaci do Java Byte Code pomocí javac kompilátoru, Dalvik VM tento byte code nespouští přímo. Java Byte Code je nejprve pomocí dex kompilátoru převeden do Dalvik Byte Code (Dalvik Executable) a až poté je tento Dalvik kód spuštěn pomocí Dalvik VM. Rozdíl viz obrázek 3.2. Důvodem pro překlad Java Byte Code do Dalvik Byte Code je zásadní rozdíl mezi implementací Java VM a Dalvik VM. Zatímco Java VM je stroj využívající zásobníkovou architekturu, Dalvik VM je stroj využívající architekturu registrů viz tabulka 3.1 [40], známou z reálných procesorů. Dalvik je optimalizován pro systémy s nízkou náročností

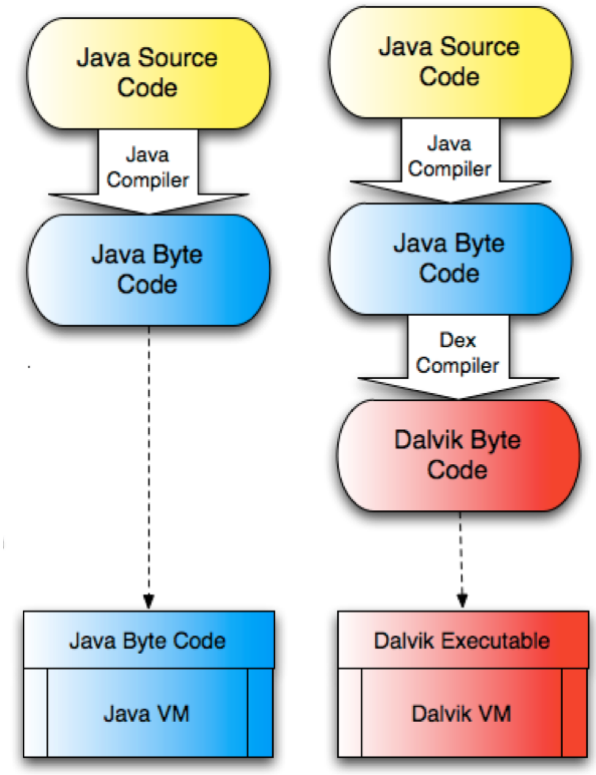


Obrázek 3.1: Android architektura

Stack-based bytekód	Register-based bytekód
iload_1	move r10, r1
iload_2	move r11, r2
iadd	iadd r10, r10, r11
istore_3	move r3, r10

Tabulka 3.1: Překlad bytekódu

na paměť. Výhodou architektury založené na registrech je potřeba menšího počtu vykonaných instrukcí v průběhu výpočtu algoritmu. Na druhou stranu zásobníková architektura umožňuje menší VM kód, takže výsledný byte kód je menší.



Obrázek 3.2: Android a Java

Pokud chceme v mobilním zařízení s OS Android volat nativní kód, musíme udělat dvě věci.

Zaprvé musíme porozumět skutečnosti, že volání nativního kódu není vhodné pro každou aplikaci a stejně tak to nemusí být vhodné ani pro nás. Volání nativního kódu s sebou přináší zvětšení komplexnosti aplikace a výsledný efekt a zrychlení nemusí být téměř žádný. Volání nativního kódu je vhodné zejména při algoritmech, které vyžadují plné zatížení procesoru a jež nealokují moc paměti, např. zpracování signálů, fyzikální simulace atd.

Druhá věc, kterou musíme udělat (v případě, že jsme si to v bodu jedna nerozmysleli) je obalit naše volání nativní funkce JNI wrapperem. JNI (Java Native Interface) je programový framework, který dovoluje Java kódu, jež běží v JVM, volat nativní kód (a být z nativního kódu volán). Dovoluje též volat knihovny napsané v jiném jazyce, jako např. v C/C++ či assembleru. Jak vytváření takového wrapperu probíhá lze vidět na obrázku 3.3 a celý proces si nyní podrobněji popíšeme.

3.1.1 Volání nativního kódu z Javy

Jedním z ideálních kandidátů pro volání nativní funkce je program pro výpočet n-tého prvku fibonacciho posloupnosti [36], jež je definovaná rekurentní rovnicí

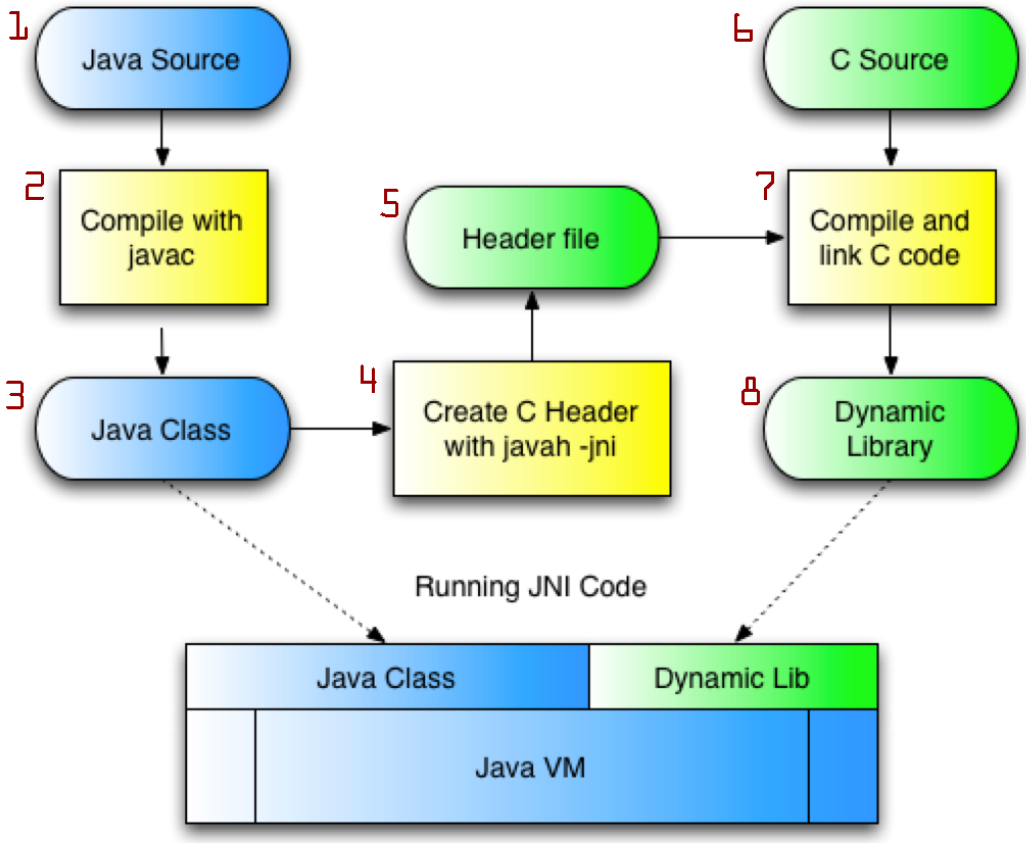
$$F_n = F_{n-1} + F_{n-2}$$

kde $F_1 = F_2 = 1$ a $F_0 = 0$. Tedy slovně řečeno každý prvek posloupnosti je roven součtu předchozích dvou prvků posloupnosti (počínaje třetím prvkem posloupnosti).

Řekněme, že chceme napsat program pro ukázku rychlosti běhu nativního kódu v porovnání s java kódem na příkladu naivního rekurzivního algoritmu výpočtu n-tého člena fibonacciho posloupnosti.

V javě nám pro to stačí jednoduchá metoda, která může být definovaná například takto:

```
public static int fibJ(int n) {
  if (n <= 0) return 0;
  if (n == 1) return 1;
  return fibJ(n - 1) + fibJ(n - 2);
```



Obrázek 3.3: Spuštění JNI kódu

```
}
```

Nyní chceme v našem programu porovnat dobu běhu této metody pro dané n s dobou běhu funkce v nativním kódu pro to samé n. Proto musíme, jak bylo řečeno výše, obalit volání nativní funkce JNI wrapperem. První věc, kterou pro to musíme udělat, je deklarovat signaturu metody v našem java programu (bod 1 na obrázku 3.3). To můžeme udělat následovně (pro větší srozumitelnost uvádíme kompletní přepis java třídy):

```
package cz.vlach.berusky;
```

```
public class FibLib {

    public native static long fibN(long n);

}
```

Klíčovým prvkem je modifikátor `native`, který říká, že se jedná o volání nativní funkce. Takto definovanou metodu můžeme z našeho java programu normálně volat, jako by se jednalo o kteroukoliv jinou metodu a s výsledkem naložit jak si přejeme. Rozdíl oproti jiným metodám je v tom, že vykonání této metody zařídí JNI v nativním kódu.

Nyní zkompilujeme třídu FibLib.java pomocí javac kompilátoru do byte kódu javy (bod 2 na obrázku 3.3). Touto kompilací dostaneme soubor FibLib.class (bod 3 na obrázku 3.3) v následující adresářové struktuře.

```
classes
└── cz
    └── vlach
        └── berusky
            └── FibLib.class
```

Nyní ze souboru FibLib.class pomocí příkazu *javah* vygenerujeme hlavičkový soubor jazyka C. Program javah je standardní součástí JDK. Vygenerování hlavičkového souboru docílíme tímto příkazem (bod 4 na obrázku 3.3):

```
[vlach@macos:classes]$ javah -jni cz.vlach.berusky.FibLib
```

Při vykonání tohoto příkazu vygeneruje program javah v aktuálním adresáři hlavičkový soubor s názvem cz_vlach_berusky_FibLib.h (bod 5 na obrázku 3.3), takže naše adresářová struktura bude vypadat takto:

```
classes
└── cz
    ├── vlach
    │   └── berusky
    │       └── FibLib.class
    └── cz_vlach_berusky_FibLib.h
```

Jméno hlavičkového souboru a jeho vnitřní struktura jsou odvozeny ze jména třídy. V případě, že tato třída je umístěna uvnitř balíčku, jméno tohoto balíčku je přidáno jak před jméno třídy, tak před jména struktur uvnitř souboru. Podtržítka (_) jsou použita jako oddělovače.

Implicitně vytváří program javah hlavičkový soubor pro každou třídu jmenovanou na příkazové řádce a vytváří hlavičkové soubory v aktuálním adresáři.

V našem případě je obsah hlavičkového souboru cz_vlach_berusky_FibLib.h následující:

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class cz_vlach_berusky_FibLib */

#ifndef _Included_cz_vlach_berusky_FibLib
#define _Included_cz_vlach_berusky_FibLib
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     cz_vlach_berusky_FibLib
 * Method:    fib
 * Signature: (J)J
 */
JNIEXPORT jlong JNICALL Java_cz_vlach_berusky_FibLib_fibN
  (JNIEnv *, jclass, jlong);

#ifdef __cplusplus
}
#endif
#endif
```

Vedle includu hlavičkového souboru `jni.h` je z hlediska další implementace nejdůležitější signatura metody `Java_cz_vlach_berusky_FibLib_fib`, kterou je nutné implementovat. První parametr metody datového typu `JNIEnv` je ukazatel na rozhraní JVM, jež obsahuje všechny funkce potřebné pro interakci s JVM a pro práci s java objekty. Druhý parametr, který je datového typu `jclass`, je objekt třídy obsahující naší metodu. O objekt třídy se jedná

Nativní datový typ	Datový typ v Javě	Popis
unsigned char	jboolean	8 bitů, bez znaménka
signed char	jbyte	8 bitů, se znaménkem
unsigned short	jchar	16 bitů, bez znaménka
short	jshort	16 bitů, se znaménkem
long	jint	32 bitů, se znaménkem
long long __int64	jlong	64 bitů, se znaménkem
float	jfloat	32 bitů
double	jdouble	64 bitů
void		

Tabulka 3.2: Předávání dat

z toho důvodu, že v javě jsme metodu `fibN(long n)` označili modifikátorem `static` a tudíž se jedná o metodu třídy a nikoliv metodu instance. V případě, že by metoda `fibN(long n)` nebyla statická tj. neobsahovala ve své deklaraci modifikátor `static`, vypadala by její signatura v hlavičkovém souboru takto:

```
JNIEXPORT jlong JNICALL Java_cz_vlach_berusky_FibLib_fib
  (JNIEnv *, jobject, jlong);
```

Tedy druhým parametrem by nebyl datový typ `jclass`, ale jednalo by se o datový typ `jobject`. Datový typ `jobject` by byl tomto případě objekt instance třídy `FibLib`.

Posledním parametrem (v obou uvedených případech) je parametr datového typu `jlong`. Jedná se o JNI ekvivalent primitivního datového typu `long`. V našem případě se jedná o jediný parametr `n` metody `fibN(long n)`.

JNI poskytuje ekvivalenty pro všechny primitivní datové typy javy, a proto lze s primitivními datovými typy pracovat v nativním kódu přímo, viz tabulka 3.2.

Objektové datové typy (tj. neprimitivní datové typy) jsou předávány jako ukazatele na vnitřní datové struktury virtuálního stroje. Uspořádání těchto struktur, ale zůstává nativnímu programu skryto. Jediný způsob, jak s těmito daty smysluplně manipulovat, je pomocí metod, které rozhraní JNI poskytuje. Ty jsou přístupné přes ukazatel `JNIEnv`.

V tomto momentě máme k dispozici hlavičkový soubor se signaturou me-

tody v jazyce C, takže zbývá pouze tuto metodu v jazyce C implementovat (bod 6 na obrázku 3.3).

```
#include "cz_vlach_berusky_FibLib.h"

jint fibN(jlong n) {
  if(n<=0) return 0;
  if(n==1) return 1;
  return fibN(n-1) + fibN(n-2);
}

JNIEXPORT jlong JNICALL Java_cz_vlach_berusky_FibLib_fibN
  (JNIEnv *env, jclass obj, jlong  n) {
  return fibN(n);
}
```

Nyní stačí tento zdrojový soubor zkompilovat (bod 7 na obrázku 3.3) a vytvořit dynamickou knihovnu (bod 8 na obrázku 3.3). S těmito kroky nám pomůže NDK, obsahující odpovídající toolchain s potřebnými nástroji.

Z obrázku 3.3 to zcela nevyplývá, ale když z javy voláme nativní kód, tak tento kód nevoláme přímo, ale prostřednictvím Java VM. Veškeré volání metod za nás udělá Java VM, my pouze v našem programu řekneme, že chceme zavolat metodu s danou signaturou. Aby Java VM mohla zavolat požadovanou funkci, potřebuje mít k dispozici potřebnou dynamicky linkovanou knihovnu (tj. knihovnu s příponou .dll na Windows, .so na Unixu nebo .jnilib na Mac OS). Pro její nahrání do našeho java programu využijeme programový blok `static { ... }`, jež je zavolán na úplném počátku spouštění našeho programu ještě před `main` metodou. Pro samotné nahrání dynamické knihovny máme k dispozici několik možností.

Mezi nejjednodušší způsoby nahrání dynamické knihovny patří využití metody `System.loadLibrary(String libname)`. JVM se pokusí nalézt knihovnu jménem `libname` v adresářích obsažených v systémové proměnné **java.library.path**. Tuto proměnnou je možné nastavit na příkazovém řádku pomocí volby `-D`. V případě, že danou knihovnu JVM nenalezne, vyhodí výjimku `java.lang.UnsatisfiedLinkError`.

Další z možností, jak nahrát dynamicky linkovanou knihovnu, je metoda `System.load(String libname)`. Od metody `System.loadLibrary(String`

`libname)` se liší v tom, že nehledá knihovnu pomocí systémové proměnné **java.library.path**, ale parametr `libname` musí být v tomto případě absolutní cestou k souboru v patřičném souborovém systému. V případě chybné cesty bude vyhozena stejná výjimka jako v předchozím případě.

Obě dvě možnosti jsou de facto ekvivalenty k volání metod `Runtime.getRuntime().loadLibrary(libname)` respektive `Runtime.getRuntime().load(libname)`.

Naše třída `FibLib` bude v kompletní podobě vypadat následovně:

```
package cz.vlach.berusky;

public class FibLib {

    public native long fibN(long n);

    static {
        System.loadLibrary("cz_vlach_berusky_FibLib");
    }
}
```

Nyní můžeme použít třídy `FibLib` a pomocí její nativní metody `fibN` a metody `fibL` uvedené na straně 26, napsat jednoduchý program který změří dobu běhu těchto dvou metod pro zadané n a porovnat rozdíl ve výkonu.

Tímto jsme pokryli ale jen jednu část problému, konkrétně volání nativní funkce z java programu. Co když ale potřebujeme jít opačným směrem? Co když chceme z nativní funkce zavolat nějakou metodu v javě? Představme si například teplotní senzor, co snímá venkovní teplotu a při její změně chceme poslat informaci o změně do java programu. Jak tohoto docílíme, se dozvíme v následující sekci.

3.1.2 Volání Javy z nativního kódu

Volání Javy z nativního kódu docílíme opět pomocí JNI (Java Native Interface) a již dříve zmíněného datového typu `JNIEnv`. Z hlediska implementace se jedná o ukazatel na strukturu obsahující všechny JNI ukazatele na posky-

tované funkce (tedy ukazatel na ukazatele), který je v hlavičkovém souboru `jni.h` definován takto:

```
typedef const struct JNINativeInterface *JNIEnv;
```

Každá z funkcí obsažená v `JNINativeInterface` je přístupná s pevným offsetem od počátku této struktury. Celkový počet funkcí se pohybuje kolem 230 ve verzi JNI 1.6 [23]. Účel těchto funkcí lze rozdělit do několika kategorií (pro větší srozumitelnost ponecháváme bez překladu):

- Version Information
- Class Operations
- Exceptions
- Global and Local References
- Weak Global References
- Object Operations
- Accessing Fields of Objects
- Calling Instance Methods
- Accessing Static Fields
- Calling Static Methods
- String Operations
- Array Operations
- Registering Native Methods
- Monitor Operations
- NIO Support
- Reflection Support
- Java VM Interface

Konkrétní názvy funkcí z uvedených kategorií zde vypisovat nebudeme, ty nejzajímavější použijeme v následujícím příkladu. Každopádně z názvů kategorií funkcí je zřejmé, že z našeho nativního programu můžeme, pomocí poskytovaným funkcím, celkem pohodlně provádět s Java VM libovolné operace.

Scénářů volání javy z nativních jazyků C/C++ je několik, od možnosti inicializace JVM kompletně z našeho nativního programu, až po pouhé zpětné volání javy z nativní funkce, kterou jsme z javy předtím zavolali. Všechny scénáře zde rozebírat nebudeme, popíšeme si pouze ten nejjednodušší scénář volání javy z nativního kódu a to je druhý uvedený scénář - zpětné volání javy z nativní funkce, kterou jsme předtím z javy zavolali.

Pro příklad viz následující výpis hlavičkového souboru a jeho implementace.

```
/* DO NOT EDIT THIS FILE - it is machine generated */
#include <jni.h>
/* Header for class Callbacks */

#ifndef _Included_Callbacks
#define _Included_Callbacks
#ifdef __cplusplus
extern "C" {
#endif
/*
 * Class:     Callbacks
 * Method:    nativeMethod
 * Signature: (I)V
 */
JNIEXPORT void JNICALL Java_Callbacks_nativeMethod
  (JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif


#include <stdio.h>
#include <jni.h>
```

```
#include "Callbacks.h"

JNIEXPORT void JNICALL
Java_Callbacks_nativeMethod(JNIEnv *env, jobject obj)
{
  jclass cls = (*env)->GetObjectClass(env, obj);
  jmethodID mid = (*env)->GetMethodID(env, cls, "callback", "(J)V");
  if (mid == 0)
    return;
  printf("In C, about to enter Java\n");
  (*env)->CallVoidMethod(env, obj, mid);
  printf("In C, back from Java\n");
}
```

Pro volání instanční metody musíme udělat následující:

1. V nativním kódu zavolat JNI funkci `GetObjectClass`, která vrací java třídu paremetru `obj`.
2. Další volanou JNI funkcí v nativním kódu je `GetMethodID`, která provede vyhledání zadané java metody v dané třídě. Vyhledání je založeno jak na jménu metody, tak na její signatuře. Pokud metoda neexistuje, vrátí `GetMethodID` hodnotu nula (0).
3. Jako poslední krok zavoláme v nativním kódu JNI funkci `CallVoidMethod`. Funkce `CallVoidMethod` zavolá instanční metodu, která má `void` jako návratovou hodnotu. Argumenty této funkce budou rozhraní JVM `env`, objekt `obj`, ID metody `mid` a případně další argumenty, které se stanou skutečnými argumenty volané funkce.

3.1.3 Tvoření názvů metod a signatur metod

JNI vyhledává metody v závislosti na názvu metody a její signatuře. Toto zajišťuje, že ta samá nativní metoda bude pracovat i poté co dojde k přidání nových metod do odpovídající java třídy.

Jméno metody je jméno metody v jazyce java v UTF-8 formátu. Konstruktor třídy specifikujeme uzavřením slova *init* do lomených závorek (což vypadá takto "`<init>`").

Signatura typu	Datový typ jazyka Java
Z	boolean
B	byte
C	char
S	short
I	int
J	long
F	float
D	double
L plně-kvalifikovaná-třída	plně-kvalifikovaná-třída
`[` *typ*	*typ* `[]`
`(typy-argumentů)návratový-typ`	typ metody

Tabulka 3.3: Předávání dat

Důležitým prvkem je seznámení se s tím, že JNI používá signatury metod pro označení návratových typů java metod. Například signatura `I(V)` značí java metodu, která má jeden parametr typu `int` a její návratový typ je `void`. Obecná podoba signatury návratové hodnoty metod je následující:

```
(typy-argumentů)návratový-typ
```

Tabulka číslo 3.3 shrnuje kódování pro signatury všech typů jazyka java.

Například metoda `Prompt.getLine` má následující signaturu:

```
(Ljava/lang/String;)Ljava/lang/String;
```

`Prompt.getLine` obsahuje jeden parametr typu `String` a návratový typ je též typu `String`.

Pole jsou indikovány počáteční hranatou závorkou (`[`), kterou následuje typ elementů pole.

Pro získání signatur metod pro použití v JNI je doporučeno využívat programu *javap* jež je součástí JDK.

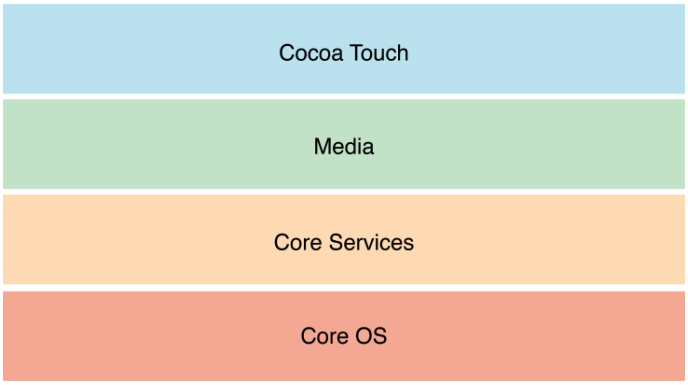
3.2 Popis architektury iOS

iOS je mobilní operační systém, který běží na zařízeních iPhone, iPod touch a iPad. Operační systém řídí hardware zařízení a poskytuje prostředky potřebné pro implementaci nativních aplikací.

iOS Software Development Kit (SDK) obsahuje prostředky a nástroje potřebné pro vývoj, instalaci, testování a běh nativních aplikací, které jsou dostupné z domovské obrazovky. Nativní aplikace jsou vytvářeny pomocí systémových frameworků iOS a programovacího jazyka Objective-C a běží přímo v operačním systému iOS. Vedle nativních aplikací je možné vytvářet webové aplikace využívající kombinaci HTML, kaskádových stylů (CSS) a JavaScriptu. Tyto aplikace na rozdíl od nativních nejsou nainstalovány přímo v iOS zařízení, ale využívají pro svůj běh webového prohlížeče Safari a vyžadují přístup na internet.

3.2.1 Architektura iOS

Architekturu operačního systému iOS lze zjednodušeně rozdělit do čtyř vrstev, viz obrázek 3.4. Nyní si krátce přiblížíme roli každé z těchto vrstev.



Obrázek 3.4: Architektura operačního systému iOS

Core OS Layer

Nejspodnější a nejzákladnější vrstvu této architektury tvoří nízko úrovňové služby, na kterých je většina vyšších vrstev postavena. I když se v aplikacích služby z této vrstvy nepoužívají, je velice pravděpodobné, že se využívají prostřednictvím frameworků z vyšších vrstev.

Součástí této vrstvy jsou frameworky poskytující přístup k externím zařízením, připojených přes datový konektor či bezdrátově pomocí Bluetooth, dále poskytuje bezpečnostní služby definované v RFC 2743 [1] a RFC 4401 [4] a v neposlední řadě poskytuje abstrakci nad samotných jádrem operačního systému.

Jádro operačního systému iOS je založeno na jádru Mach [26], jež poskytuje nízko úrovňový přístup k jádru pomocí UNIXových rozhraní. Tyto rozhraní poskytují podporu pro:

- Vlákna (POSIX threads)
- Sítě (BSD sockets)
- Přístup k souborovému systému
- Standardní I/O
- Správu paměti

Core Services Layer

Tato vrstva poskytuje základní systémové služby, které jsou využívány všemi aplikacemi.

Mezi tyto základní služby patří:

iCloud Storage umožňuje aplikacím ukládat uživatelské dokumenty a data do centrálního úložiště a přistupovat k nim ze všech iOS zařízení uživatele.

Automatic Reference Counting jedná se o funkci na kompilační úrovni, jež se stará o uvolňování paměťových prostředků alokovaných naší aplikací.

In-App Purchase poskytuje možnost nákupu obsahu či služeb uvnitř naší aplikace.

SQLite dovoluje do naší aplikace vložit SQL databázi bez nutnosti běhu vzdáleného databázového serveru.

XML Support obsahuje základní prvky pro manipulaci s XML dokumentem.

Media Layer

Tato vrstva obsahuje technologie pro práci s grafikou, videem a zvukem.

Mezi základní knihovny poskytované touto vrstvou patří:

Core Graphics též známá pod názvem Quartz, je knihovna pro renderování obrázků a 2D vektorové grafiky.

Core Animation poskytuje pokročilou podporu pro tvorbu komplexních animací a vizuálních efektů.

Core Image poskytuje silnou podporu pro manipulaci s videem a statickými obrázky od jednoduchých operací, až například po podporu detekce obličejů i jiných tvarů.

OpenGL ES poskytuje podporu pro 2D a 3D renderování pomocí hardwarové akcelerace.

Core Text poskytuje sofistikovaný engine pro sazbu a renderování textů.

Vedle těchto služeb nabízí tato vrstva ještě další služby jako AirPlay, Assets knihovnu a další. Jedná se však o služby přesahující rozsah této práce.

Cocoa Touch Layer

Tato vrstva obsahuje klíčové frameworky pro vývoj iOS aplikací jako například multitasking, dotykový vstup, notifikace a další vysoko úrovňové systémové služby.

Mezi služby poskytované touto vrstvou patří:

Auto Layout pomocí pravidel definujeme rozložení elementů uživatelského rozhraní, jedná se o vylepšení klasického "springs and struts" modelu.

Storyboards umožnuje navrhnout celé uživatelské rozhraní na jednom místě, takže je možno vidět pohromadě všechny pohledy a řadiče i s jejich vzájemnou interakcí.

Multitasking při zmáčknutí Home tlačítka, není v popředí běžící aplikace ukončena, ale její běh je přesunut do pozadí. Tato služba podporuje uspání takové aplikace pro prodloužení životnosti baterie.

Printing tato službu umožnuje aplikacím tisk na bezdrátové tiskárně.

Push Notification poskytuje možnost, jak upozornit uživatele na nové informace i přesto, že naše aplikace není aktivní.

Gesture Recognizers jedná se o objekty, které jsou propojeny s pohledem naší aplikace a využívány pro detekci různých typů gest, jako tažení prstem, poklepání prstem atd.

Co se týče běhu C/C++ kódu na platformě iOS, tak zde je situace podstatně jednodušší oproti platformě Android. Hlavním programovacím jazykem platformy iOS je jazyk Objective-C [31], což je rozšíření jazyka C o systém zasílání zpráv z jazyka Smalltalk. Díky této skutečnosti je použití kódu napsaného v jazyce C v programu napsaném v jazyce Objective-C zcela bezproblémové. Bylo by dokonce možné se jazyku Objective-C vyhnout úplně, ale vzhledem k tomu, že většina výše zmíněných služeb systému iOS je napsaná v Objective-C, tak minimálně část kódu aplikace sloužící k interakci s okolím je vhodné v tomto jazyce napsat.

3.3 Zhodnocení platforem

V předchozích dvou sekcích jsme ukázali, že předmět této práce, konverze počítačové hry na platformy iOS a Android, tak aby co největší část práce byla společná pro obě platformy, je splnitelný. Z popisu jednotlivých platforem vyplývá, že zejména pro platformu Android takový převod bude vyžadovat o něco více práce.

4 Realizační část

V této části si podíváme na to, co všechno obnášela konverze hry Berušky na mobilní zařízení.

4.1 Vývojové prostředí pro Android

Pro vývoj pro platformu Android potřebujeme tyto tři věci **Vývojové IDE**, **Android SDK** a **Android NDK**.

Vývojových IDE pro Android je dnes celá řada jako například Eclipse, IntelliJ IDEA či NetBeans, takže si každý může vybrat podle svých preferencí. Oficiálně podporovaným IDE je však pouze Eclipse IDE a tak si ukážeme, jak právě toto IDE nakonfigurovat.

Jelikož zdrojový kód Berušek je napsán v jazyce C++ a tuto základnu kódu chceme v maximální možné míře využít, budeme potřebovat Eclipse s podporou jazyka C/C++. Stáhneme si tedy tyto nástroje:

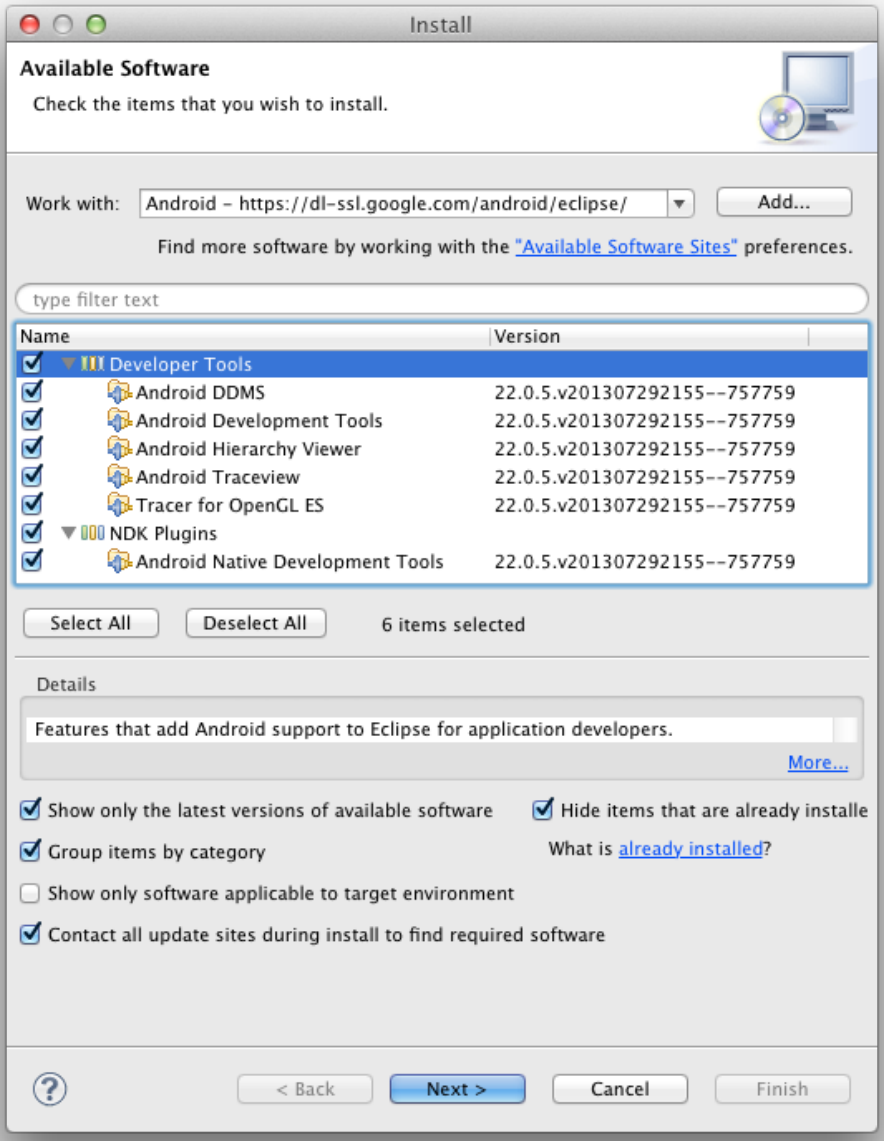
Eclipse IDE for C/C++ Developers ze stránek `http://www.eclipse.org`

Android SDK ze stránek `http://developer.android.com/sdk`

Android NDK ze stránek `http://developer.android.com/tools/sdk/ndk`

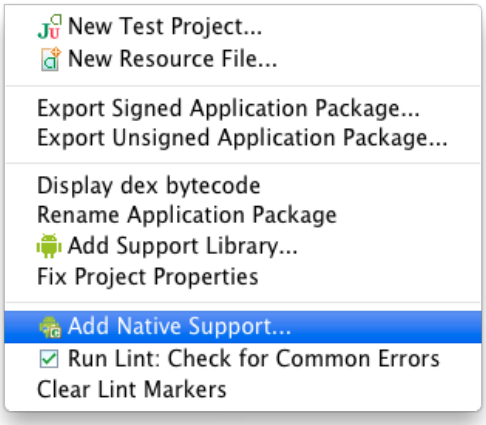
Po rozbalení archivu s Eclipse IDE a jeho spuštění je nutné do této Eclipse instance nahrát potřebné pluginy podporující vývoj na platformu Android, jež se standardně v Eclipse IDE nenacházejí. Jak vypadá instalace těchto pluginů viz obrázek 4.1, kde se nachází i Eclipse update site s těmito pluginy, více informací viz [35].

Dalším krokem je rozbalení SDK a NDK do adresářů dle vlastní volby a nakonfigurování Eclipse IDE kde tyto adresáře nalezne. Odpovídající volba pro SDK se nachází zde: `Preferences -> Android -> SDK Location`, pro



Obrázek 4.1: Instalace podpory Androidu do prostředí Eclipse

NDK se nachází zde: `Preferences -> Android -> Native Development -> NDK Location`.



Obrázek 4.2: Přidání podpory nativního kódu do Android projektu

Nyní již můžeme v Eclipse IDE založit nový projekt `File -> New -> Project.. -> Android -> Android Application Project`. Po projití průvodce založením nového projektu máme v našem pracovním adresáři založen nový Android projekt. Tento projekt zatím nemá podporu nativního kódu. Podporu NDK do vytvořeného projektu přidáme kliknutím na projekt pravým tlačítkem myší a vybráním volby `Android Tools -> Add Native Support..` viz obrázek 4.2. Nyní je náš projekt připraven ke spuštění nativního kódu.

Hlavním rozdílem mezi projektem s podporou nativního kódu a projektem bez této podpory je v přítomnosti adresáře `jni` v adresářové struktuře projektu s nativní podporou. Obsahem adresáře `jni` budou naše zdrojové soubory v jazyce C/C++ a build soubor s názvem `Android.mk`. V tomto build souboru vypíšeme C/C++ soubory s naším nativním kódem a Eclipse IDE se automaticky postará o jejich zkompilování a sestavení.

Jelikož ve složce `jni` budou kromě zdrojových kódů hry Berušky též zdrojové kódy samotné knihovny SDL2, je nutné umístit každý z nich do vlastního podadresáře. Umístíme tedy zdrojové kódy hry Berušky do složky `jni/src`. Knihovnu SDL2 nahrajeme na adresáře `jni/SDL` (knihovna SDL2 již obsahuje odpovídající `Android.mk` soubor pro zkompilování a sestavení).

Nyní je projekt téměř připraven pro spuštění hry Berušky na zařízení Android a následné úpravy pro potřeby tohoto mobilního zařízení. Jediné co schází udělat je upravit kód z SDL 1.2 knihovny na SDL 2.0 knihovnu. Tyto

úpravy jsou popsány v sekci 4.4.1.

4.2 Vývojové prostředí pro iOS

Při vývoji pro platformu iOS máme život jednodušší v tom, že nemusíme přemýšlet nad tím jaké IDE zvolit, protože můžeme zvolit jediné IDE a tím je vývojový nástroj Xcode od společnosti Apple. Toto IDE je k dispozici pouze na počítačích Apple Mac, je tedy nutné si počítač této řady pořídit.

Instalaci nástroje Xcode, je nutné provést skrze Apple Store a je zcela bezproblémová. Výhoda (či nevýhoda) vývoje v Xcode je to, že toto IDE nás nenutí k jakékoliv předepsané struktuře projektu, jakékoliv členění souborů projektu je plně v naší režii. Této vlastnosti s úspěchem využijeme v sekci 4.3 kde se budeme zabývat problémem sdílení zdrojového kódu pro obě dvě platformy.

Díky volnosti co nám Xcode ve struktuře projektu nabízí, ale též přichází otázka jakou strukturu projektu zvolit. Naštěstí knihovna SDL2.0 nabízí template projektu s podporou knihovny SDL2.0, takže tohoto templatu využijeme a odpadne nám tak nutnost vymýšlení vlastní struktury.

4.3 Struktura projektu pro dvě platformy/dvě IDE

Standardní součástí každého většího projektu musí být SCM systém. Nejenom, že takový systém řeší spolupráci více uživatelů na jednom projektu, ale díky verzování zdrojových souborů, se lze kdykoliv vrátit k jakékoliv verzi zdrojového kódu. Tahle vlastnost je obzvláště nedocenitelná v případech kdy je potřeba udělat rozsáhlou změnu ve zdrojových kódech (např. refaktoring), která může skončit neúspěchem a je potřeba vrátit se k původní verzi. Bez systému SCM, by toto byl těžko dosažitelný úkol. Zejména z tohoto důvodu zavedeme SCM i pro účely této práce.

Největší popularitě se v dnešní době díky svým vlastnostem těší SCM systém Git [32]. Využijeme ho proto i v našem projektu.

Problémem v našem úkolu je jak strukturovat zdrojový kód tak, aby mohl být tento zdrojový kód využíván dvěma různými IDE prostředími pro dvě různé mobilní platformy, jmenovitě prostředí Eclipse pro platformu Android a prostředí Xcode pro platformu iOS.

Pro úspěšné vyřešení našeho problému musíme učinit detailní analýzu struktur projektů pro jednotlivá IDE a možností jejich přizpůsobení našim potřebám.

4.3.1 Struktura Android projektu

Vytvoříme-li ve vývojovém prostředí Eclipse nový projekt pro Android, zjistíme pohledem na souborový systém, že projekt se skládá z následujících souborů a adresářů:

```
/project-root
        ├── /assets
        ├── /jni
        ├── /libs
        ├── /res
        ├── /src
        ├── AndroidManifest.xml
        └── project.properties
```

Krátce si popíšeme účel každého z těchto adresářů:

`project-root` kořenový adresář celého projektů (název tohoto adresáře může být libovolný),

`assets` adresář obsahující obrázky, zvuky atd.,

`jni` adresář obsahující všechen nativní kód,

`libs` adresář obsahující knihovny nutné pro běh projektu,

`res` adresář obsahující obrázky, zvuky atd.,

`src` adresář obsahující java zdrojové kódy.

Rozdíl mezi adresáři `assets` a `res` je v tom, že pro prostředky z adresáře `res` je k dispozici podpora pro více jazykových mutací, rozlišné velikosti obrazovek, orientaci obrazovek atd. Pro prostředky ze složky `assets` nic takového k dispozici není. Složka `assets` se hodí např. pro ukládání vygenerovaných obrázků za běhu aplikace atd.

Kombinace dvou programovacích jazyků Java a C++ a toho, že musí společně kooperovat a komunikovat, přináší do struktury projektu pro Android komplikace a mnoho věcí funguje na základě konvencí nežli na základě konfigurace. Z tohoto důvodu je změna struktury projektu pro Android téměř nemožná.

Zkusíme tedy vyzkoumat, co by znamenalo použít strukturu projektu pro Android pro projekt iOS.

4.3.2 Struktura iOS projektu

Projekt v Xcode má sice danou strukturu, ale tato struktura je pouze relativní, tj. neodpovídá struktuře adresářů a souborů na souborovém systému. Vše je tedy jen otázkou konfigurace.

Knihovna SDL 2.0 ve svém zdrojovém kódu navíc obsahuje template projektu využívajícího SDL, kde je již vše podstatné předkonfigurované. Použitím tohoto templatu a drobnými úpravami jsme schopný projekt pro iOS nakonfigurovat nad adresářovou strukturou diktovanou požadavky projektu pro systém Android.

4.4 Portace

V následujících kapitolách si podrobně podíváme na jednotlivé oblasti, ve kterých byli nutné řešit problémy při portaci hry Berušky na mobilní zařízení. Nejprve si popíšeme problémy spojené s novou knihovnou SDL 2.0 a jejich řešení, poté se podíváme na problém změny z velké obrazovky na malý displej a jako poslední si popíšeme změny nutné v ovládání hry.

4.4.1 Změny SDL 2.0 oproti SDL 1.2

Knihovna SDL 2.0 doznala od verze SDL 1.2 značných změn. Zde si postupně projdeme základní prvky, jak knihovna SDL 2.0 pracuje a povíme si, jaký byl původní kód hry Berušky.

První krok pro použití knihovny SDL 2.0 je její inicializace. Ta se provádí funkcí `SDL_Init`, která má jeden parametr, kterým lze říci, jaký subsystém SDL chceme v naší aplikaci využívat. Subsystém může být například video, audio, joystick apod. V našem případě provedeme pouze inicializaci subsystému video následujícím příkazem:

```
SDL_Init(SDL_INIT_VIDEO);
```

V oblasti inicializace subsystémů žádná změna v SDL 2.0 oproti SDL 1.2 nenastala, a proto ani kód hry Berušky neprošel žádnou změnou.

Další krokem při použití knihovny SDL je vytvoření okna, do kterého budeme kreslit výsledný obraz. To se provede funkcí `SDL_CreateWindow`, která obsahuje sedm parametrů. Zde je její signatura:

```
SDL_Window * SDL_CreateWindow(const char *title,
    int x, int y, int w, int h, Uint32 flags)
```

Z názvů parametrů je vidět jejich význam. Potenciál těchto parametrů lze plně využít pouze v případě klasických obrazovek. V případě mobilních zařízení zodpovědnost za podobu a velikost okna přebírá operační systém. Ten dovoluje vytvořit pouze okna přes celou obrazovku, případně okno obsahující horní lištu. V našem případě okno vytvoříme tímto příkazem:

```
int flags = SDL_WINDOW_BORDERLESS
SDL_Window* window = SDL_CreateWindow("Berusky", 0, 0, 0, 0, flags);
```

Počínaje touto funkcí začíná jedna z nejvýraznějších změn v SDL 2.0 oproti SDL 1.2. Touto změnou je rozdělení softwarového renderování a hardwarového (akcelerovaného) renderování. Toto renderování probíhalo v SDL 1.2 pomocí datové struktury `SDL_Surface`, kde hardwarové renderování bylo

zjistitelné pomocí zapnutého příznaku `SDL_HWSURFACE`. Ve verzi SDL 2.0 došlo k rozdělení softwarového a hardwarového renderování takže nyní máme k dispozici tyto dvě struktury:

`SDL_Surface` struktura použita pro softwarové renderování, není nikdy akcelerovaná,

`SDL_Texture` struktura použita pro hardwarové akcelerování, její pixely nejsou přímo přístupné.

S tímto rozdělením přišla i změna způsobu jak se vytvářejí tyto struktury pro renderování.

Druhou změnou v SDL 2.0 oproti SDL 1.2, jež je s problematikou vykreslování úzce spjata, je možnost vytváření více oken, což v SDL 1.2 možné nebylo. Každé z vytvořených oken má asociovaný renderer, což je struktura `SDL_Renderer`. Každý renderer je spravován grafickým ovladačem.

Tento renderer nahrazuje objekt `SDL_Surface` z SDL 1.2. Obsahuje několik metod pro kreslení bodů, čar, obdélníků, textur atd. Pomocí tohoto rendereru probíhá veškeré kreslení.

Vrátíme se k naší funkci `SDL_CreateWindow` pomocí které vytváříme okno pro hru Berušky. V našem případě jsme výšku `w`, šířku `h`, pozici `x`, pozici `y` nastavili na `0`, protože jak jsme si již řekli, v případě mobilních zařízení je podoba vytvořeného okna plně v režii operačního systému. Pouze pro systém iOS je potřeba explicitně říci příznakem `SDL_WINDOW_BORDERLESS`, že chceme vytvořit okno bez horní lišty.

Pro to abychom mohli v oknu zobrazit nějaký obraz, potřebujeme, jak jsme si popsali, vytvořit pro toto okno renderer a s oknem ho asociovat. To provedeme pomocí funkce `SDL_CreateRenderer`. Zde je její signatura:

```
SDL_Renderer * SDL_CreateRenderer(SDL_Window * window,
    int index, Uint32 flags)
```

Prvním argumentem je okno, pro které renderer vytváříme, druhým argumentem je index ovladače pro renderování, který chceme inicializovat nebo -1 pokud chceme inicializovat první ovladač podporující požadované příznaky. Třetím parametrem jsou příznaky renderovacího kontextu.

V našem případě vypadá konkrétní volání následovně:

```
SDL_Renderer *renderer = SDL_CreateRenderer(window, -1, 0);
```

Ve verzi knihovny SDL 1.2 byla práce těchto dvou funkcí provedena funkcí s názvem `SDL_SetVideoMode`. Odpovídající kód ze hry Berušky vypadal následovně:

```
SDL_Surface *p_hwscreen = SDL_SetVideoMode(width, height, bpp, flag);
```

Tento příkaz vytvořil okno, pomocí něhož probíhalo renderování výsledného obrazu. Toto volání a datovou strukturu `SDL_Surface` bylo potřeba v kódu hry Berušky nahradit odpovídajícím rendererem `SDL_Renderer` a provést odpovídající úpravy ve struktuře kódu.

Dalším krokem při získání finálního obrazu je nutnost načíst ze souborů herní textury. Hra Berušky má herní textury v datovém formátu bmp a toto je jediný typ obrazových souborů, který knihovna SDL dokáže načíst (pro načítání jiných typů souborů by bylo potřeba použít knihovnu SDL_image).

Obě verze SDL obsahují makro `SDL_LoadBMP` jež umožnuje načíst obrázek ze souboru ve formátu bmp. Jeho signatura spolu se signaturami funkcí, které toto makro využívá, vypadají následovně:

```
#define SDL_LoadBMP(file)
        SDL_LoadBMP_RW(SDL_RWFromFile(file, "rb"), 1)

SDL_Surface * SDL_LoadBMP_RW(SDL_RWops * src, int freesrc)

SDL_RWops * SDL_RWFromFile(const char *file, const char *mode)
```

Z uvedených signatur a názvů funkcí je celkem zřejmé co makro `SDL_LoadBMP` dělá.

Pomocí makra `SDL_LoadBMP` hra Berušky nahraje obrázek ze souboru zadaného parametrem `file` do datové struktury `SDL_Surface`. Pokud se nahrání obrázku ze souboru povede, uloží se do datové struktury `SDL_Texture`. Toto nahrání se provede pomocí funkce `SDL_CreateTextureFromSurface`, jejíž signatura vypadá následovně:

```
SDL_Texture * SDL_CreateTextureFromSurface(
    SDL_Renderer * renderer, SDL_Surface * surface)
```

Prvním parametrem je renderer, který jsme v předchozím kroku asociovali s naším oknem pro zobrazení výsledného obrazu. Druhým parametrem je datová struktura obrázku načtená ze souboru pomocí makra `SDL_LoadBMP`. V následujícím odstavci si ukážeme, jak je naprogramován tento proces ve hře Berušky.

```
// Globální proměnná
SDL_Renderer *renderer;

//-----------------------

SDL_Texture *p_text;

void surface::load(char *p_file)
{
  SDL_Surface *p_tmp = SDL_LoadBMP(file);

  if(p_tmp) {
    p_text = SDL_CreateTextureFromSurface(renderer, p_tmp);
    assert(p_text);
    SDL_FreeSurface(p_tmp);
  }
}
```

Nyní si ukážeme, jak byla tato část naprogramována ve verzi pro desktopové operační systémy za pomocí verze SDL 1.2:

```
SDL_Surface *p_surf;

void surface::load(char *p_file)
{
  SDL_Surface *p_tmp = SDL_LoadBMP(file);

  if(p_tmp) {
    p_surf = SDL_DisplayFormat(p_tmp);
```

```
    assert(p_surf);
    SDL_FreeSurface(p_tmp);
  }
}
```

V původní verzi hry Berušky jsme pomocí makra `SDL_LoadBMP` nahráli obrázek ze souboru a poté jsme pomocí funkce `SDL_DisplayFormat` zkopírovali tento obrázek do nové struktury `SDL_Surface` v pixelovém formátu a barvách grafického video framebufferu.

Jak je vidět v porovnání předešlých dvou ukázek kódu, co se týče struktury zdrojového kódu, tak nová verze se od původní verze liší zejména použitím rendereru. Z toho důvodu, aby mohl být tento renderer předán do funkce `SDL_CreateTextureFromSurface`, musel být zpřístupněn jako globální proměnná a zdrojový kód patřičně upraven.

Používání výplně

Další nutnou změnou v renderování výsledného obrazu byla nahrazení funkce `SDL_FillRect` pro vyplnění zadané obdélníkové plochy jednolitou barvou, funkcí `SDL_RenderFillRect`. Signatury těchto funkcí vypadají následovně:

```
int SDL_FillRect(SDL_Surface *dst,
  SDL_Rect *dstrect, Uint32 color);

int SDL_RenderFillRect(SDL_Renderer * renderer,
  const SDL_Rect * rect)
```

Ihned na první pohled je vidět rozdíl mezi jednotlivými verzemi knihovny SDL. Význam parametru `dstrect` v původní funkci `SDL_FillRect` je stejný jako význam parametru `rect` v nové funkci `SDL_RenderFillRect`. Jedná se o určení plochy, která se má vyplnit barvou. V případě funkce `SDL_FillRect` je barva výplně předána funkci parametrem `color`. V knihovně SDL 2.0 je nutné před voláním funkce `SDL_RenderFillRect` nastavit požadovanou barvu výplně zavoláním metody `SDL_SetRenderDrawColor`, která pro zadaný renderer nastaví barvu výplně.

Promítání obrázků - blitting

Poslední výraznou změnou v renderování výsledného obrazu byla nutnost nahrazení funkce `SDL_BlitSurface` z SDL 1.2 funkcí `SDL_RenderCopy` z SDL 2.0. Signatury obou funkcí jsou následující:

```
int SDL_BlitSurface(
  SDL_Surface *src, SDL_Rect *srcrect,
  SDL_Surface *dst, SDL_Rect *dstrect);

int SDL_RenderCopy(SDL_Renderer*   renderer,
                   SDL_Texture*    texture,
                   const SDL_Rect* srcrect,
                   const SDL_Rect* dstrect)
```

Účelem těchto dvou funkcí je promítnutí části zdrojového obrázku na cílový obrázek. V případě funkce `SDL_BlitSurface` je zdrojem parametr `src` a cílem je parametr `dst`. V případě funkce `SDL_RenderCopy` je zdrojem parametr `texture` a cílem parametr `renderer`. Parametr `srcrect` je v případě obou dvou funkcí určení části obrazu co se bude promítat a parametr `dstrect` je určení místa kam se zdrojový obrázek promítne.

I v tomto případě jsou změny ve zdrojovém kódu hry Berušky vynuceny změnou vnitřní architektury knihovny SDL 2.0 oproti knihovně SDL 1.2.

4.4.2 Uzpůsobení dotykovému ovládání

Jedním z nejsložitějších úkolů je předělat ovládání hry. Původní verze hry Berušky byla ovládána myší a klávesnicí. Hry na mobilní zařízení s sebou přinesli nový způsob ovládání, kde je veškerá interakce mezi hrou a hráčem docílena pomocí dotykového displeje a gest, kde mezi nejvíce používané gesta patří klepnutí prstem na displej a tažení prstem po displeji. Dalším velmi častým gestem je přiblížení/oddálení dvou prstů, používané pro přiblížení/oddálení herní plochy.

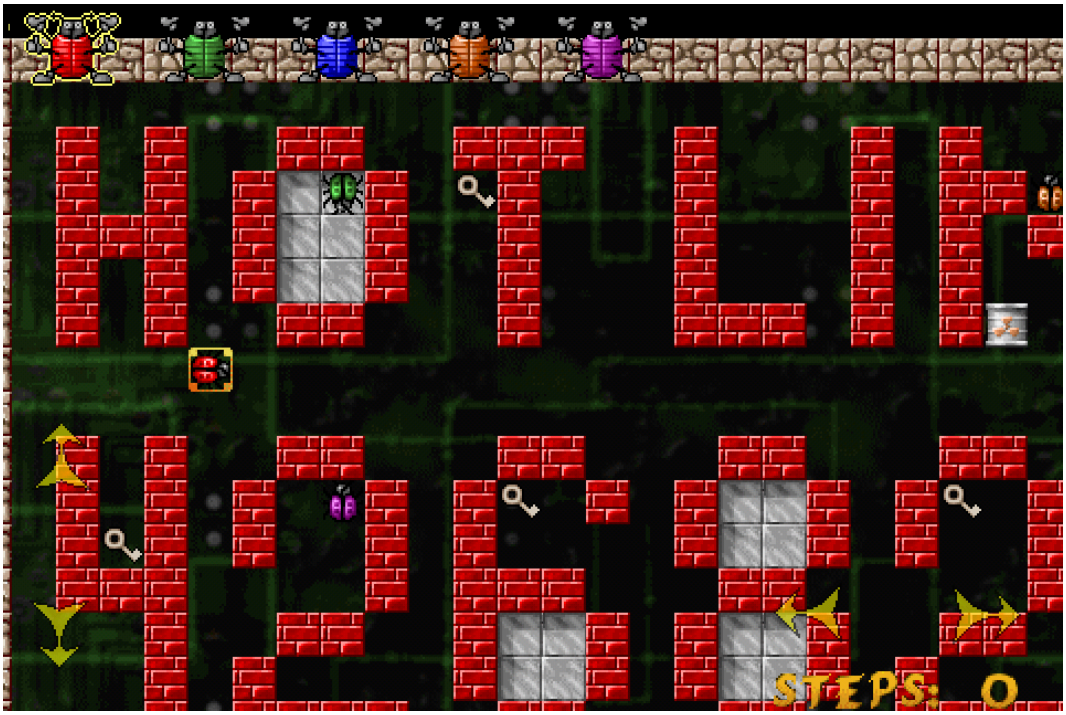
Ovládání původní hry Berušky patří naštěstí k velice jednoduchým, kde se veškeré ovládání provádí pouhými pěti klávesami (klávesy pro funkce Nápověda, Uložení/Nahrání hry nepočítáme). Klávesy šipka nahoru/dolů/doleva/

doprava slouží k pohybu berušky po herním levelu a klávesa TAB slouží k přepínání mezi beruškami. Jako indikátor jakou má hráč právě vybranou berušku, slouží lišta v horní části obrazovky. Na této liště jsou obrázky jednotlivých berušek, kde se jednotlivé beruška od sebe odlišují barvou, viz obrázek 4.3 znázorňující PC verzi hry Berušky.

Obrázek 4.3: Ukázka PC verze hry Berušky

Ačkoliv horní lišta je v případě PC verze hry Berušky neaktivní (přepínání berušek probíhá pomocí klávesy TAB), jsou obrázky berušek na ní zobrazené ideálním ovládacím prvek pro mobilní verzi hry. Využili jsme této skutečnosti a v mobilní verzi hry Berušky implementovali přepínání berušek pomocí klepnutí na jejich obrázky v horní liště obrazovky.

Po vyřešení přepínání berušek zbývá vyřešit problém ovládání pohybu berušek. Jedním možným řešením by bylo využití gyroskopického senzoru a ovládat berušky podle orientace zařízení. Nevýhodou tohoto typu ovlá-



Obrázek 4.4: Ukázka mobilní verze hry Berušky

dání je, že vyžaduje od hráče mít mobilní zařízení v určité poloze a tím ho do značné míry omezuje. Využijeme tedy klasické ovládání, kde vyhrazené oblasti obrazovky mobilního zařízení slouží jako náhrada kláves šipka nahoru/dolů/doleva/doprava. Otázkou zůstává jak tyto oblasti (klávesy) na obrazovce rozmístit. Jednou možností je zůstat u klasického rozložení kláves, kdy klávesa nahoru a klávesa dolů jsou mezi klávesami doleva a doprava. Toto rozložení dovoluje ovládání pohybu berušky pouze jednou rukou, ale na druhou stranu zde hrozí poměrně veliký počet přehmatů. Druhou možností je rozdělit ovládání na dvě části a seskupit klávesy nahoru a dolů na jednu stranu obrazovky a na druhou stranu obrazovky umístit klávesy doleva a doprava. Tento styl ovládání neumožnuje pohodlné ovládání jednou rukou, ale na druhou stranu je toto ovládání velice přesné a nehrozí při něm téměř žádná možnost přehmatu. Navíc toto ovládání je velice blízké ovládání z klasických gamepadů herních konzolí, kde většina ovládání probíhá pomocí palců na obou rukách. Z tohoto důvodu jsme se rozhodli právě tento typ ovládání implementovat i do mobilní verze hry Berušky.

Pro pohyb berušky dolů je vyhrazena levá spodní oblast obrazovky, pro

pohyb nahoru je vyhrazena oblast těsně nad oblastí pro pohyb dolů. Donutit berušku k pohybu doprava je možné t'uknutím (a držením) v pravé spodní oblasti obrazovky, pohyb beruškou doleva je umístěn vlevo od oblasti pro pohyb doprava. Pro zvýraznění tohoto ovládání je obrazovka doplněna čtveřicí průhledných šipek umístěných v oblastech pro konkrétní pohyb beruškou viz obrázek 4.5.

Co se týče samotné implementace změny ovládání, snažili jsme na úrovni zdrojového kódu do maximální míry využít stávající systém ovládání, abychom do něho museli zasáhnout co nejméně. Nejtěžší na tomto úkolu bylo pochopit logiku kódu implementujícího toto ovládání. Ve zjednodušené podobě si lze logiky ovládání představit jako nekonečnou smyčku herních událostí (kde událostí může být stisk/puštění klávesy/obrazovky) a při každé z této herní události se porovná zdroj této události s polem klávesových-událostí. Toto pole klávesových-událostí je ve zdrojovém kódu hry Berušky natvrdo předdefinováno a říká, co se má pro jakou herní událost stát. Existence tohoto pole klávesových-událostí nám dovoluje do kódu hry Berušky jednoduše přidat metodu, která bude převádět t'uknutí na obrazovku na zmáčknutí klávesy. Jak tato funkce vypadá, viz následující kód:

```
#define UP      0
#define DOWN    1
#define LEFT    2
#define RIGHT   3

#define PLAYER_0   9
#define PLAYER_1   10
#define PLAYER_2   11
#define PLAYER_3   12
#define PLAYER_4   13
#define MENU_IN_GAME   14

int input::get_pressed_key(SDL_Event * event) {

  float mx = ((float) event->tfinger.x);
  float my = ((float) event->tfinger.y);

  if(mx < 0.3 && my > 0.5) {
      return my > 0.8 ? DOWN : UP;
  }
```

```
  if(mx > 0.7 && my > 0.6) {
      return mx > 0.85 ? RIGHT : LEFT;
  }

  if(mx > 0.9 && my < 0.1) {
      return MENU_IN_GAME; // pravý horní roh obrazovky
  }

  if(my < 0.16) {
    float x = 120.0/game_resolution_x;
    if(mx < x) {
        return PLAYER_0;
    }
    if(mx > x     && mx < x * 2) {
        return PLAYER_1;
    }
    if(mx > x * 2 && mx < x * 3) {
        return PLAYER_2;
    }
    if(mx > x * 3 && mx < x * 4) {
        return PLAYER_3;
    }
    if(mx > x * 4 && mx < x * 5) {
        return PLAYER_4;
    }
  }
  return -1;
}
```

Čísla konstant v ukázce kódu jsou indexy do pole klávesových-událostí. Toto pole se při každé herní události celé projíždí a odpovídající klávesové-události se vykonávají. Z ukázky kódu je zároveň vidět, že t'uknutím do pravého horního rohu obrazovky se hráč dostane do herního menu.

Další oblastí, kterou bylo nutné kompletně předělat je výběr herních levelů. V původní verzi hry Berušky bylo přecházení mezi levely vyřešeno pomocí hesel, kdy se hráči po dohrání každé úrovně zobrazilo heslo pro přístup do další úrovně. Toto heslo si bylo nutné opsat a při dalším spuštění hry bylo nutné toto heslo zadat do odpovídající kolonku a zmáčknou tlačítko Play, aby se hráč dostal do požadovaného levelu. Tento přístup je pro mobilní hru

Obrázek 4.5: Ukázka výběru levelu v mobilní hře Berušky

zcela nevhodný a naprosto odporuje zaběhnutým standardům na poli mobilních her. Jak již bylo řečeno v kapitole 2.1 obsahuje hra Berušky celkem 120 herních úrovní. Tyto úrovně jsou rozděleny celkem do pěti obtížností, přičemž nejvíce úrovní obsahuje obtížnost Easy a to rovných 50.

V mobilních hrách je standardem, že hry, které jsou postaveny na velkém množství krátkých levelů, např. hra Angry Birds [12], nabízí hráči obrazovku, na které je seznam více levelů najednou a hráč si může zvolit, jaký level chce (samozřejmě nechybí princip postupného odemykání levelů jednoho po druhém). Navíc je typické, že hra nabízí více levelů, nežli se vejde do seznamu na jednu obrazovku, a proto existuje se seznamem levelů více obrazovek. Mezi těmito obrazovkami se obecně pohybuje gestem tažením prstu. A právě takový systém jsme implementovali do mobilní verze hry Berušky, viz obrázek 4.5. Na obrázku je zachycen okamžik přechodu mezi první a druhou obrazovkou s výběrem levelů. Tato část hry byla oproti původní verzi hry Berušky kompletně předělána právě pro potřeby mobilních zařízení.

4.4.3 Změna z obrazovky na malý displej

Problémy při portaci hry Berušky popsané v předešlé kapitole měli všechno co dočinění s ovládáním hry. V této kapitole si povíme o problémech, které bylo při portaci nutné vyřešit, spojené s velikostí obrazovky mobilních zařízení.

Prvním problémem, který se s velikostí obrazovek mobilních zařízení objevil, je paradoxně jejich vysoké rozlišení. V rámci boje o zákazníka se výrobci mobilních zařízení snaží všemožnými silami býti lepší než konkurence a do tohoto boje spadá i honba za největším rozlišením. Původní hra Berušky běžela v rozlišení 640 x 480 pixelů, což s přehledem dnešní mobilní telefony překonávají. Problémem je, že v tak *malém* rozlišení je hra Berušky na displeji mobilního telefonu velice malá a například na zařízení s dnes běžným rozlišením obrazovky 1196 x 768 zabírá necelou polovinu obrazu.

Programový kód byl z důvodu malého rozlišení původní hry upraven tak, že v momentě kdy se pomocí funkce `SDL_RenderCopy`, popsané v sekci 4.4.1, promítá zdrojový obrázek na cílový, tak se hodnoty souřadnic se kterými se pracuje, násobí číslem 2. Toto zajistí, že výsledný obraz je dostatečně velký a využívá celkovou plochu obrazovky i u zařízení s velkým rozlišením.

Tato úprava s sebou nese zároveň i další problém. Ačkoliv máme nyní celou obrazovku mobilního zařízení využitou, nové rozlišení 1280 x 960 je v poměru 4:3 (jako původní rozlišení), ale širokoúhlý displej mobilního zařízení takový poměr stran nenabízí, a proto se část herní plochy kreslí mimo obrazovku. Tento problém je ještě více patrný například na mobilním zařízení Apple iPhone 4S, jež disponuje rozlišením 960 x 640 pixelů. V tomto případě se mimo obrazovku kreslí již podstatná část herní plochy.

Tento problém je možné vyřešit pouze jediným způsobem. Tím způsobem je umožnění hráči posouvat hrací plochou dle jeho libosti. Samotnou implementaci řešení tohoto problému lze již provést několika způsoby a lze tuto možnost posouvání hrací plochy rozšířit o možnost přiblížení/oddálení (zoom) hrací plochy. Tato kombinace nabízí nejvíce uživatelského komfortu, ale za cenu složitějšího ovládání a komplikovanější implementace. Jelikož by možnost zoomování přinesla další otázky, na které není jednoznačná odpověď, jako např. jak moc umožnit hráči herní plochu přiblížit/oddálit atd., rozhodli jsme se od možnosti zoomování upustit a zaměřili jsme se pouze na možnost pohybování hrací plochou bez změny přiblížení.

Samotná možnost posunutí hrací plochy gestem tažení prstem sebou nese

několik otázek, které je nutné zvážit. Nejdůležitější z těchto otázek je jak daleko může hráč hrací plochu posunout. Bude moci hráč posunout hrací plochou mimo hranice obrazovky? A pokud ano, co se stane, když tažení přeruší? Co se zobrazí v prostoru mimo hrací oblast atd.? Abychom se vyhnuli těmto otázkám, implementovali jsme posunutí hrací plochy pouze v rámci hranic obrazovky. To znamená, že pokud hráč táhne hrací plochou vpravo, povolíme mu táhnout maximálně tak daleko, aby na levém okraji obrazovky nevznikla žádná mezera mezi okrajem obrazovky a okrajem herní plochy. Výjimkou z tohoto pravidla je tažení herní plochy dolů. V tomto případě hráči dolovíme táhnout tak daleko, že horní okraj hrací plochy se posune směrem dolů pod horní lištu zobrazující ikony berušek. Pro lepší představu viz obrázek C.1, zobrazující ve své horní polovině stav, kdy je herní plocha mimo lištu s beruškami (obrazovka obsahuje horní část hrací plochy) a v dolní polovině je obrázek, zobrazující situaci, kdy je herní plocha posunuta nahoru, takže hráč vidí spodní část hrací plochy a lišta s beruškami se kreslí nad hrací plochou.

Obrázek 4.6: Ukázka PC verze hry Berušky

Dalším problémem úzce souvisejícím s nutností posouvat hrací plochou, z důvodu rozlišných velikostí displejů mobilních zařízení, je jakým způsobem zobrazovat na obrazovce herní informace. Mezi tyto informace patří počet kroků, které berušky ušli od začátku herní úrovně, počet sebraných klíčů a počet sebraných krumpáčů, které jednotlivé berušky vlastní. Všechny tyto informace byly v původní verzi hry zobrazeny nad hrací plochou nebo pod ní a byli stále vidět. V mobilní verzi hry Berušky bylo, pro zobrazení těchto informací, zapotřebí začít rozlišovat lokální souřadnice v herní úrovni a globální

souřadnice obrazovky, které zůstávají po celou dobu hry stejné.

Další problém, jenž je nutný v souvislosti s vykreslováním herní úrovně vyřešit, je situace kdy hráč pohybuje beruškou a dostane se s ní k hranicím obrazovky v momentě, kdy herní plocha pokračuje dále. Pokud by se nic nestalo, tak by se v dalším kroku beruška dostala do části hrací plochy mimo obrazovku a hráč by v tu chvíli nevěděl, co beruška právě dělá, takže by mohla omylem posunout nějakou bednu či podobně. Takové chování by mělo negativní dopad na spokojenost hráče, a proto jsme do mobilní verze hry Berušky implementovali automatické posunutí hrací plochy v momentě, kdy se právě ovládaná beruška přiblíží k okraji obrazovky a směrem, kterým se beruška pohybuje je herní plocha mimo obrazovku. Tento systém dovoluje projít celou herní úroveň bez nutnosti ručního posouvání hrací plochy.

Poslední problém velice úzce souvisí s automatickým posouváním hrací plochy, přiblíží-li se beruška hranici obrazovky. Tímto problémem je, že v některých situacích kdy hráč přepíná jednotlivé berušky, tak nově vybraná beruška je mimo viditelnou hrací plochu. V tomto případě chce hráč ihned vědět, kde se nachází jím zvolená beruška. Tento problém je vyřešen automatickým přepnutím na část hrací plochy obsahující nově vybranou berušku.

5 Závěr

Cílem této práce bylo zkonvertovat počítačovou hru Berušky na mobilní platformy Android a iOS a prozkoumat možnosti běhu C/C++ kódu na těchto platformách. V této práci jsme si nejdříve krátce představili obě dvě platformy iOS a Android a poté jsme se podívali na možnosti multiplatformního vývoje aplikací pomocí dostupných řešení. V této části práce jsme zjistili, že existuje široký výběr z technologií a frameworků, od těch poskytujících možnost psaní aplikací pomocí HTML5 a CSS, až po klasické frameworky poskytující možnost psaní aplikací v jazyce C/C++. Ve vyhodnocení této části jsme se rozhodli, pro účel konverze využít knihovnu SDL 2.0.

V další části práce jsme se blíže podívali na možnosti běhu C/C++ kódu na jednotlivých platformách a u platformy Android jsme si podrobně popsali jak běhu C/C++ kódu docílit, jelikož primárním programovacím jazykem této platformy je jazyk Java.

V realizační části této práce jsme si popsali vývojové prostředí pro jednotlivé platformy a struktury projektů, jež tyto vývojová prostředí předepisují. Zjistili jsme, že projekt pro platformu Android musí mít jasně danou strukturu a využívá se zde konvencí před konfigurací. Na druhé straně od projektu pro platformu iOS není vyžadována žádná striktní struktura. Této skutečnosti jsme s výhodou využili při sdílení jednoho zdrojového kódu pro obě platformy verzovacím systémem Git.

V druhé polovině realizační části práce jsme si podrobně popsali problémy při převodu počítačové hry na mobilní zařízení s dotykovým ovládáním. Nejprve jsme prošli nutné změny ve zdrojovém kódu s ohledem na vykreslování výsledného obrazu, poté jsme si prošli implementaci dotykového ovládání a v poslední části jsme si řekli něco o problémech spojených s rozlišením obrazovek mobilních zařízení.

Cíl této práce se nám podařilo naplnit a nyní lze počítačovou hru Berušky hrát na zařízeních s operačním systémem iOS a Android.

A Seznam použitých zkratek

HTML HyperText Markup Language

CSS Cascading Style Sheets

GPU Graphics processing unit

API Application Programming Interface

RAD Rapid application development

IDE Integrated Development Environment

NDK Native Development Kit

VM Virtual Machine

JVM Java Virtual Machine

JNI Java Native Interface

JDK Java Development Kit

SDL Simple DirectMedia Layer

RFC Request for Comments

XML Extensible Markup Language

SCM Source Control Management

B Instalační příručka

Na přiloženém CD naleznete kompletní Git repositář, jež obsahuje zdrojové soubory hry Berušky a zároveň konfiguraci Eclipse projektu i Xcode projektu. Tyto konfigurační soubory lze využít pro import projektu hry berušky do těchto vývojových prostředí.

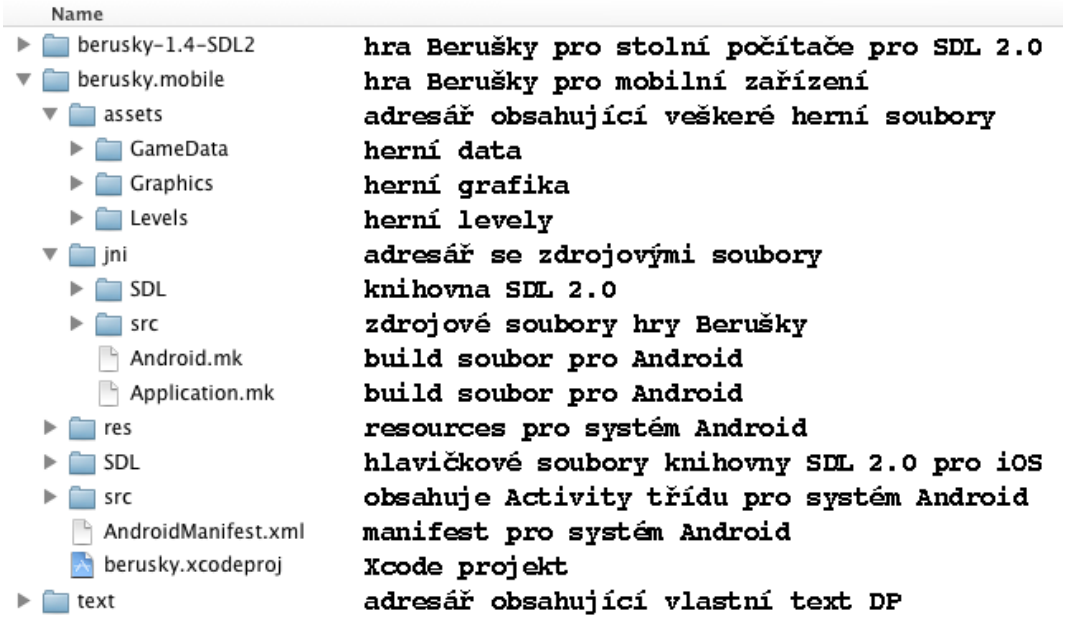
Pro spuštění na Android zařízení stačí požadované zařízení připojit pomocí USB a v prostředí Eclipse spustit hru Berušky na tomto zařízení.

Pro spuštění na iOS zařízení je potřeba stát se členem iOS Developer Programu. Členství v tomto programu je placené. Po zaplacení členství je potřeba nakonfigurovat prostředí Xcode, což v sobě zahrnuje vytvoření vlastního certifikátu a dalších věcí. Podrobné informace lze nalézt na oficiálních stránkách iOS Developer Programu.

Na přiloženém CD se v adresáři `berusky.mobile/jni/SDL` nachází dodatečné informace, jak zprovoznit projekt obsahující SDL na jednotlivých platformách.

C Obsah přiloženého CD

Přiložené CD obsahuje kompletní Git repositář použitý během implementační části této práce. Na obrázku je popis nejdůležitějších adresářů a souborů.



Obrázek C.1: Obsah přiloženého CD

Literatura

[1] RFC 2743. http://www.ietf.org/rfc/rfc2743.txt, Srpen 2013.

[2] Generic LEvel EDitor 2D. http://gleed2d.codeplex.com, Srpen 2013.

[3] Stage 3D. http://www.adobe.com/devnet/flashplayer/stage3d.html, Srpen 2013.

[4] RFC 4401. http://tools.ietf.org/html/rfc4401, Srpen 2013.

[5] Adobe Air. http://dojotoolkit.org/features/mobile, Srpen 2013.

[6] Open Handset Alliance. http://www.openhandsetalliance.com, Srpen 2013.

[7] Anakreon. http://anakreon.cz, Srpen 2013.

[8] 975 Apple patent č.7, 786. http://www.google.com/patents/US7786975, Srpen 2013.

[9] Apportable. http://www.apportable.com, Srpen 2013.

[10] Badland. http://www.badlandgame.com, Srpen 2013.

[11] Android bagatelizován. http://www.engadget.com/2007/11/05/symbian-nokia-microsoft-and-apple-downplay-android-relevance, Srpen 2013.

[12] Angry Birds. http://www.angrybirds.com, Srpen 2013.

[13] Box2D. http://box2d.org, Srpen 2013.

[14] CocosBuilder. http://cocosbuilder.com, Srpen 2013.

[15] Apache Cordova. http://cordova.apache.org, Srpen 2013.

[16] Berušky domovská stránka. http://anakreon.cz/?q=node/1, Srpen 2013.

[17] Tiled Map Editor. http://www.mapeditor.org, Srpen 2013.

[18] Away3d Game Engine. http://away3d.com, Srpen 2013.

[19] Citrus Game Engine. http://citrusengine.com, Srpen 2013.

[20] Nape Physics Engine. http://napephys.com, Srpen 2013.

[21] OpenGL ES. http://www.khronos.org/opengles/, Srpen 2013.

[22] Oznámení frameworku SDL 2.0. http://www.phoronix.com/scan.php?page=news_item&px=MTE0MDU, Srpen 2013.

[23] Specifikace Java Native Interface. http://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/jniTOC.html, Srpen 2013.

[24] Sokoban je PSPACE-úplný. http://webdocs.cs.ualberta.ca/~joe/Preprints/Sokoban, Srpen 2013.

[25] jQuery Mobile. http://jquerymobile.com, Srpen 2013.

[26] Mach Kernel. https://en.wikipedia.org/wiki/Mach_(kernel), Srpen 2013.

[27] Simple DirectMedia Layer. http://www.libsdl.org, Srpen 2013.

[28] Lua. http://www.lua.org, Srpen 2013.

[29] Dojo Mobile. http://www.adobe.com/products/air.html, Srpen 2013.

[30] Rozložení mobilního trhu. http://marketingland.com/mobile-market-share-android-iphone-gain-while-others-flail-50197, Srpen 2013.

[31] Objective-C. http://en.wikipedia.org/wiki/Objective-C, Srpen 2013.

[32] Git Home Page. http://git-scm.com/, Srpen 2013.

[33] Away Physics. http://www.awayphysics.com, Srpen 2013.

[34] Bad Piggies. http://www.badpiggies.com, Srpen 2013.

[35] Eclipse Android Plugin. http://developer.android.com/sdk/installing/installing-adt.html, Srpen 2013.

[36] Fibonacciho posloupnost. http://en.wikipedia.org/wiki/Fibonacci_number, Srpen 2013.

[37] Cut The Rope. http://www.cuttherope.ie, Srpen 2013.

[38] Corona SDK. http://www.coronalabs.com, Srpen 2013.

[39] Marmalade SDK. http://www.madewithmarmalade.com, Srpen 2013.

[40] Yunhe Shi, David Gregg, Andrew Beatty, and M. Anton Ertl. Virtual machine showdown: stack versus registers. In *In VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, pages 153–163. ACM, 2005.

[41] Právní spor Goolge vs. Oracle. http://www.wired.com/business/2010/10/google-oracle-android, Srpen 2013.

[42] Starling. http://gamua.com/starling, Srpen 2013.

[43] Sencha Touch. http://www.sencha.com/products/touch, Srpen 2013.

[44] Historie verzí operačního systému iOS. http://en.wikipedia.org/wiki/IOS_version_history, Srpen 2013.

[45] Dalvik VM. http://en.wikipedia.org/wiki/Dalvik_(software), Srpen 2013.

[46] Unity zdarma. http://blogs.unity3d.com/2013/05/21/putting-the-power-of-unity-in-the-hands-of-every-mobile-,year={Srpen2013}developer.

[47] zlib/libpng License (Zlib). http://opensource.org/licenses/Zlib, Srpen 2013.