Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra počítačových systémů

Bakalářská práce

Middleware pro vývoj aplikací pro děti

Vojtěch Dvořák

Vedoucí práce: Ing. Jiří Smítka

24. března 2015

Poděkování

Děkuji především přítelkyni Tereze a celé rodině za psychickou podporu v době psaní této práce a panu Ing. Jiřímu Smítkovi za odborné rady.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 24. března 2015 .....

Odkaz na tuto práci

Dvořák, Vojtěch. *Middleware pro vývoj aplikací pro děti*. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2015.

Abstrakt

Tato práce si klade za cíl navrhnout a implementovat middleware, tzv. mezivrstvu, která by i méně zkušeným programátorům dokázala jednoduše a rychle umožnit vytváření výukových multimediálních a multiplatformních aplikací pro děti. Součástí práce je také ukázkové nastavení centrálního distribučního serveru, který umožní hromadnou správu vytvořených aplikací.

Klíčová slova Middleware, mezivrstva, multimédia, síťování, výuka, děti

Abstract

The goal of this thesis is to design and create programmer's middleware for easy and fast development of educational, multimedia and multiplatform applications for kids. The thesis also includes example settings of central distribution server which allows remote control of deployed applications created with middleware.

Keywords Middleware, multimedia, networking, education, kids

Obsah

Seznam obrázků

Úvod

S rychlým vývojem v oblasti informačních technologií a komplexní digitalizací většiny každodenních akcí, je snižován průměrný věk uživatelů, kteří začínají pravidelně používat počítače a jiná elektronická zařízení. Řeč je především o dětech předškolního a školního věku. Naší povinností je řádně připravit budoucí generace na digitální éru a zábavnou formou je naučit výpočetní zařízení používat a zároveň upevnit jejich znalosti v jiných odvětvích jako je matematika, logika, biologie a další. Jak praví jedno známé přísloví: „Žádný učený z nebe nespadl.“ Jakou formu výuky by bylo nejvýhodnější zvolit?

Děti jsou hravé, zvědavé a také jsou známy jako jedno z nejnáročnějších publik, co se týče nároků na obrazový a zvukový doprovod v aplikaci. Animace, barevné obrázky, mluvící zvířátka, efekty, jednoduše multimediální kolos se vším všudy. Je třeba děti motivovat do učení, ukázat jim, že i vzdělávání může být zábava. Při prvním seznámení s počítačem bývá pro jakéhokoli nového uživatele největší překážkou polohovací zařízení, v našem případě myš. Stejně je tomu u seniorů, kteří zaspali počítačový věk, jako u dětí prvně postavených před výzvu v podobě nového zařízení. Při samotném návrhu komponent pro výslednou realizaci middlewaru jsem se soustředil především na komponenty zajišťující výuku motorických schopností mladých uživatelů.

Zadání tématu bakalářské práce ovšem nevede přímo k realizaci hotové aplikace pro výuku dětí, nýbrž nástroje pro vývojáře, na kterém by i ne-příliš zkušený programátor dokázal pro děti realizovat dříve zmíněné body, při vykonání co nejmenšího objemu práce. Může se jednat stejně tak o učitele jako o zaneprázdněného rodiče, který chce své potomky alespoň na chvíli zabavit.

Pro potřeby výuky v počítačových učebnách je vhodné zamyslet se nad hromadnou distribucí a správou vytvořených aplikací. Ve světě, kdy je drtivá většina všech počítačů zapojena do nějaké sítě, se před námi otevírá široká nabídka cest, kterými se jako realizátoři můžeme vydat.

Kapitola 1

Cíl práce

Cílem práce je návrh a následná implementace middleware pro programátory, který umožní jednoduché vytváření výukových multimediálních a multiplatformních aplikací pro děti. Takto vytvořené aplikace musí podporovat obrázky, zvuky, animace a interakci s uživatelem za pomoci myši. Výsledné programy vytvořené middlewarem bude možné nasadit na počítače s různými operačními systémy a budou jednoduše instalovatelné.

Pro potřeby škol bude middleware obsahovat podporu síťové komunikace na úrovni stahování podpůrných souborů z centrálního distribučního serveru pomocí HTTP protokolu [23]. Součástí práce bude také ukázková konfigurace zmíněného serveru, založeného na open-source technologiích (vhodných pro nasazení ve výukových zařízeních).

Kapitola 2

Analýza

Tato kapitola si klade za cíl analýzu existujících řešení a sběr dat pro budoucí vývoj. Rozeberu zde konkurenční programy (viz 2.1), v sekci 2.2 možnosti programovacích jazyků, jejich nadstaveb a knihoven, které by bylo možné pro realizaci middlewaru použít a nakonec ještě provedu rešerši týkající se požadavků na zaměření budoucích aplikací v sekci 2.3.

Omezující podmínkou pro výběr existujícího řešení je především požadavek na multiplatformnost, tedy možnost nasazení výsledného programu na více než jednu platformu operačního systému. Při hodnocení použitelnosti konkurenčních řešení pro tvorbu multimediálních aplikací je třeba vzít v potaz také to, zda tyto programy umožňují programátorovi „rozvoj“, popřípadě „vývoj“ aplikace šité na míru.

2.1 Existující programy

Pro začátek by bylo vhodné upozornit na fakt, že samotné téma bakalářské práce je svým způsobem unikátní, neboť je velmi úzce zaměřeno a vymezeno. Budeme se v této sekci věnovat řešením, jejichž cílovou skupinou jsou děti předškolního a školního věku. To je samo o sobě velmi omezující podmínka. Dále je třeba zhodnotit faktor modifikovatelnosti nalezených výukových nebo multimediálních programů.

Middleware je velmi obecné pojmenování jakési mezivrstvy, ať už se jedná o formu programů typu server-klient, konfigurovatelné běhové prostředí nebo samotné frameworky. Vše zmíněné se dá považovat za middleware. Chtěl bych vybrat zástupce jednotlivých kategorií, probrat jejich klady a zápory.

2.1.1 Programy s minimální konfigurovatelností

Programů pro výuku dětí je ohromné množství, komerčních i nekomerčních, ale postrádají základní požadavek a tím je konfigurovatelnost. Ve většině případů jsou tématicky zaměřeny pouze na jednu oblast a stávají se tak jedno-

účelovými. Takový program bývá jednou nainstalován, třikrát spuštěn a opět odebrán ze systému. Tím nepoukazuji na špatnou kvalitu těchto programů. Tyto aplikace podlehly monopolistickému trendu a jsou ve velké míře distribuovány pouze pro operační systém Windows. Sluší se uvést i nejznámější zástupce této skupiny:

- Všeználek
- Matematika 1.0
- Žáček 2.1
- spousty dalších ...

2.1.2 Hry na nových technologiích

V této podkapitole bych se rád zmínil především o možnostech, které přinesl nástup nových technologií s dotykovým ovládáním. Řeč je především o tabletech a chytrých mobilních telefonech. Lavina aplikací, která se na uživatele odevšad valí, je obrovská. Mezi nimi jsou početně zastoupeny právě různé hry a aplikace výukové, určené především pro děti. Ačkoliv jsem se lehce odchýlil od původního tématu konfigurovatelných aplikací a aplikací určených především na osobní počítače, sluší se vyjmenovat některé nejznámější zástupce i této oblasti:

- Kids ABC Train Game (Intellijoy)
- DR. Panda's restaurant (TribePlay)
- Omalovánky (Playground)
- Cut the rope (ZeptoLab)

2.1.3 Existující řešení odpovídající zadání

Jak jsem již dříve zmínil, zadání samo o sobě je velmi specifické a při hledání existujících middleware pro vývoj aplikací pro děti, které by splňovaly všechny požadavky nebo se funkcionalitou k vytyčenému řešení alespoň přibližovaly, jsem byl neúspěšný.

2.2 Frameworky a nadstavby programovacích jazyků

Po neúspěšném hledání výukového programu, který by byl více konfigurovatelný a multiplatformní, jsme nuceni se ponořit hlouběji do oblastí vyžadujících alespoň nějakou úroveň programovacích schopností člověka, který aplikace

pro děti bude sestavovat.

S frameworkem neboli nadstavbou programovacího jazyka se v moderním světě technologií potkáváme čím dál častěji. Účel je jediný: usnadnit práci vývojářům softwaru pomocí zobecnění programovacích postupů, zapouzdření typických a opakovaných činností při psaní kódu. Frameworky řeší ten aspekt tvorby, který bývá v dnešním světě IT jedním z nejdůležitějších a tím je časová náročnost implementace. Bývají odladěné a tudíž bezpečné a námitka, že provádění kódu je pomalejší, se stává zanedbatelnou v závislosti na technologické vyspělosti firem, které každým rokem chrlí čím dál tím více výkonnější hardware. Výsledek je navíc zapsán v přehledné formě a ve většině případů umožňuje pokročilejší techniky ladění.

Ve frameworku lze vytvářet i aplikace pro děti, je však nutné říci, že nástroje, které zde budu jmenovat, nejsou přímo zaměřeny na tvorbu výukových programů, tedy implementace tímto způsobem vyžaduje mimo odborných znalostí i hodně času.

Pro účely vytváření GUI a multimediálních aplikací jmenujme jako nejznámější nadstavbu Swing (viz 2.2.1) programovacího jazyka Java a Qt (viz 2.2.2) pro C++. Důležité je, že obě zmíněné nadstavby jsou distribuovány jako open-source.

Seznam nejznámějších frameworků

- Nette pro PHP
- .NET pro C#
- jQuery pro JavaScript

2.2.1 Swing

Nadstavba jednoho z nejpoužívanějších jazyků Java je populární pro tvorbu uživatelského rozhraní neboli GUI. Jedná se o framework využívající architekturu MVC. Poskytuje možnosti tvorby jednodušších aplikačních celků a her. Samozřejmou vlastností je podpora multi-threadingu, tedy běhu v režimu více vláken. Je vhodným kandidátem pro tvorbu algoritmicky méně náročných programů, jako jsou šachy, piškvorky a zjednodušené vizualizace, především z důvodu nepříliš svižného běhového prostředí Java.

2.2.2 Qt

Co znamená Swing pro Javu, tím je Qt pro programovací jazyk C++. Knihovna je stejně jako Swing multiplatformní a dovoluje tvorbu složitějších uživatel-

ských rozhraní. Velkým plusem pro tuto knihovnu je fakt, že ji můžeme nalézt v implementacích různých programovacích a skriptovacích jazyků:

- C++
- Python
- Ruby
- C
- Pascal
- Java

2.3 Dětské obecenstvo

Před návrhem testovacích aplikací jsem se snažil ujasnit si jakousi vizi toho, jak by výsledné programy měly vypadat. Tvořit aplikace pro děti, jste-li dospělým, může být tvrdým oříškem, neboť vnímání podnětů těchto dvou skupin se celkem zásadně liší. V rešerši této oblasti mi pomohlo především přečtení knihy K. Bäcker-braun, *Rozvoj inteligence u dětí od 3 do 6ti let* [2].

Prozkoumání této oblasti bylo důležité k finalizaci požadavků na middleware, jakou funkcionalitu by měl obsahovat a na co bych se měl při tvorbě zaměřit. Výsledky mého zkoumání jsem adaptoval na použití ve výukových hrách a shrnul do následujících bodů:

2.3.1 Oblasti, na které se u dětí zaměřit

- Procvičování paměti
- Matematické schopnosti a logika
- Prostorová a plošná představivost
- Postřeh a rychlost myšlení
- Fantazie, tvořivost
- Práce s písmeny a čtení

2.3.2 Příklady her a aktivit rozvíjející tyto schopnosti

- Pexeso
- Počítání
- Omalovánky

- Doplňování písmen do slov
- Bludiště
- Spojovačky

2.3.3 Funkcionalita nutná pro realizaci

- Barvy
- Pohyb
- Zvuky
- Interakce

Oblastí, na kterou se naopak zaměřovat vůbec nebudu, popřípadě jen okrajově, je rozvoj verbálních dovedností dítěte. Například implementace rozpoznávání řeči je velmi náročná a dokázala by obsáhnout samostatné téma závěrečné práce.

Kapitola 3

Návrh

Při hledání existujícího řešení v kapitole 2, které by odpovídalo zadání, jsem byl neúspěšný. Rozhodl jsem se tedy vytvořit middleware vlastní, který by zahrnul všechny požadované aspekty tvorby multimediálních a multiplatformních aplikací pro děti.

V této kapitole se budu věnovat návrhu samotného middlewaru. Osvětlím důvody pro výběr určitého realizačního jazyka, v sekci 3.1, přiblížím postupy, kterými bych se chtěl řídit při tvorbě (viz 3.2) a nastíním představu technologií, použitých pro centrální distribuční server v sekci 3.3.

3.1 Výběr realizačního jazyka

Při výběru jazyka pro realizaci middleware jsem zvažoval různé kombinace. Zamýšlel jsem se nad otázkou typu stroje, na kterém by měly výsledné aplikace být spustitelné. Vezmeme-li v úvahu, že programy budou vyučovat nejmenší z žáků základních škol, popřípadě předškoláky, narážíme na palčivý problém technické specifikace počítače.

Mnoho škol si nemůže dovolit nakupovat drahé počítačové vybavení a strach učitelů a učitelek z poškození přístrojů dětmi je myslím si v těchto případech oprávněný. Školy, které nedokáží dávat finanční prostředky na zajištění základních učebních pomůcek pro děti, si těžko dovolí kupovat licence na operační systémy s dnes tak používanými Windows a sáhnou raději po open-source řešení ve formě některé linuxové distribuce.

3.1.1 Skriptovací jazyky

Hned ze začátku jsem zavrhnul použití skriptovacích jazyků především kvůli jejich dopadu na výsledný výkon aplikace.

3.1.2 Java

Velmi oblíbený a moderní programovací jazyk Java splňuje požadavek multiplatformnosti, jelikož aplikace běží ve virtualizovaném prostředí a toto prostředí je distribuováno jak na OS Windows tak i na Linux. Při spojení s některou z multimediálních knihoven by skutečně bylo možné uvažovat nad nasazením tohoto řešení. Narážíme zde ale na výkonostní problém, který by mohl nastat při spouštění takto vytvořených programů na starších strojích.

Mým osobním názorem zůstává, že použití běhového prostředí Java na složitější multimediální aplikace není nejšťastnější. Java se také dlouhodobě potýká s bezpečnostními problémy a tak i když naše aplikace nemusí být zabezpečeny jako bankovní, může tento aspekt použití JVM přidat pomyslný hřebíček do rakve.

3.1.3 C/C++

Nízkoúrovňový programovací jazyk, který na rozdíl od Javy nevyžaduje speciální běhové prostředí. Při použití portovatelných knihoven (např. SDL, boost) lze vytvářet velmi svižné a multiplatformní aplikace.

Díky jeho elementárnosti jsem se nakonec rozhodl jej v realizaci bakalářské práce použít. Z velkého množství knihoven pro C++ [4] jsem vybral následující:

- SDL (Simple Directmedia Layer, verze 1.2)
- SDL_image 1.2 (práce s obrázky)
- SDL_mixer 1.2 (ovládání zvukových zařízení)
- SDL_ttf 1.2 (knihovna pro vykreslování ttf fontů)
- boost [21] (objemná multiplatformní knihovna obsahující všemožné funkce)
- Standard Template Library (STL [11])

Nespornou výhodou rozšiřujících modulů SDL je široký záběr jejich souborových formátů. Díky nim můžeme do aplikace načítat všemožné multimediální formáty: BPM, GIF, JPEG, LBM, PCX, PNG, PNM, TGA, TIFF, WEBP, XCF, XPM, XV, WAV, OGG, MIDI a spousty dalších. Nejsme tedy limitováni převodem souborů poskytnutých programátorem do nějaké programem stravitelnější podoby.

3.1.3.1 SDL

SDL [24] neboli *Simple DirectMedia Layer*. Pomocí funkcí tohoto souboru je možné vytvářet vlastní multimediální a především multiplatformní aplikace. Samotné SDL v základu obsahuje pouze sadu elementárních funkcí pro práci

s grafikou a zvuky. Existují ale i navazující projekty (SDL_image, SDL_mixer, ...), které rozšiřují funkcionalitu SDL. Samozřejmě se jedná o framework portovatelný jak na operační systémy Linux tak i Windows a projekt jako takový je open-source. Zahrnuje podporu vstupních zařízení (myš, klávesnice, joystick) a vykreslování pomocí OpenGL a Direct3D. Je tedy vhodným kandidátem pro tvorbu multiplatformních aplikací a her. Nativní jazyk SDL je C++. Existují však i verze pro Python a C#. Díky skvělým návodům na rootu [26] jsem se celkem rychle dokázal v SDL zorientovat.

3.2 Architektura aplikace

Jak již bylo dříve zmíněno, middleware skýtá spousty realizačních možností. Pro mou práci jsem se rozhodl jít cestou spouštěného běhového prostředí, kterému pomocí konfiguračního souboru programátor zadá vlastnosti, jak se má celá aplikace chovat. Pro tyto účely jsem se rozhodl vytvořit nový popisovací jazyk, který by byl srozumitelný pro každého, i neprogramátora, a pomocí kterého by byly dosažitelné všechny požadované funkce výsledné aplikace.

Celý middleware bude podléhat objektovému návrhu, neboť používám programovací jazyk C++. Konfigurační soubor zapsaný ve vlastním jazyce (viz. 3.2.1) bude běhovému prostředí předán při startu přes parametr, popřípadě bude stažen z centrálního distribučního serveru.

3.2.1 Vlastní popisovací jazyk

Soubor s direktivami bude načten jednou při startu a bude moci odkazovat na multimediální materiál, umístěný na filesystému (obrázky, hudbu, ...). Jazyk bude umožňovat složitější vztahy mezi jednotlivými rámy aplikace, odkazovat se na ně. Direktivy a vztahy budou zapsány anglicky, jelikož je to standart v oblasti informačních technologií. Popisovat bude možné jednotlivé komponenty, jejich vlastnosti a celkové chování.

3.2.2 Komponenty

Pro účely výuky dětí jsem se rozhodl implementovat jednotlivé skupiny tématicky zaměřených komponent. Tyto komponenty budou navržené tak, aby byly obecněji použitelné.

3.3 Technologie distribučního serveru

V našem případě si distribuční server můžeme představit jako učitele, který svým žákům (klientům) poskytuje učební pomůcky, předčítá jim z knih a říká, jak se mají chovat. Komunikace mezi nimi bude probíhat pomocí TCP/IP a HTTP protokolu.

3.3.1 Operační systém

Výběr operačního systému a softwaru na distribuční server byl ovlivněn především mírou globálního rozšíření a oblíbeností u uživatelů/správců. Na ukázkovém stroji nasadím operační systém Linux s distribucí Debian [1].

3.3.2 HTTP server

Komunikace mezi klientem (middlewarem) a distribučním serverem bude probíhat na dnes nejrozšířenějším přenosovém protokolu, kterým je *HyperText Transfer Protocol*.

Nejznámějším a zároveň nejnasazovanějším HTTP serverem je celosvětově projekt HTTPD Apache [25] (k dnešnímu datu ve verzi 2.6) a ten jsem tudíž ve své práci použil. Apache má široké možnosti konfigurace a podporuje načítání všelijakých modulů. Server bude poslouchat na HTTP standartním portu 80 a poskytovat ke stažení soubory konfigurace vytvořené aplikace a podpůrné soubory nutné k běhu (hudbu, obrázky, fonty, ...). Poté co dojde k úspěšnému stažení stromové struktury projektu klientem, bude komunikace se serverem ukončena.

Kapitola 4

Realizace middlewaru

V této kapitole naleznete popis vlastní realizace. Implementuji middleware pro vývoj aplikací pro děti v programovacím jazyce C++. Pro produkční účely jsem vybral název *Donkey*. Použiji nadstavbové knihovny multimediálních funkcí SDL a Boost, které umožní vytvořit portabilní software. Celý kód je nutně psán multiplatformně, tudíž se vyhýbám knihovnám, které nejsou přenosné mezi operačními systémy. Middleware bude fungovat na principu běhového prostředí, které umožní načítání konfiguračních souborů zapsaných ve speciálním popisovacím jazyce, relativně jednoduše editovatelných běžným uživatelem. Cílem je implementace funkčního konfigurovatelného prostředí, které bude navíc komunikovat pomocí síťového protokolu HTTP s centrálním distribučním serverem. Zdrojový kód je bohatě komentován pomocí značek nástroje *doxygen* [7].

4.1 Architektura

Middleware podléhá objektovému návrhu. Architektura je spíše jednodušší variantou MVC (Model-view-controler) řízenou přes události. V kódu je použita dědičnost, především do první úrovně. Třídy jako takové se dají rozdělit do tří základních skupin. První z nich je hlavní třída CDonkey (viz 4.2.1), která zapouzdřuje celý middleware do jednoho celku. Na hlavní třídu navazují komponenty obecné (viz 4.3) a speciálně navrhnuté pro potřeby výuky dětí (viz 4.4). Jako modul jsou připojeny CHTTP (viz 4.6) a CDonkeyParser (viz 4.7) třídy, které zajišťují jakousi nadstavbu základního rámce.

4.2 Základní a rodičovské třídy

Třídy, které obalují funkcionalitu tříd podřízených. Nejdůležitější rodičovskou jednotkou je zde třída CDonkey a pak CControl, která popisuje obecné principy funkcionality komponenty.

4.2.1 CDonkey

Jedná se o základní třídu celého běhového prostředí middlewaru (neobsahuje síťování ani parser). Při naimportování zdrojů a hlavičkových souborů lze pomocí volání vnitřních funkcí vytvořit multimediální aplikaci přímo v kódu C++ (bez použití parseru a konfiguračních souborů) stylem psaní stejným jako při tvorbě frameworky. Ukázka vytvoření nejjednodušší aplikace přímo v kódu (tato varianta není doporučená - API je určeno pro postavení aplikace voláním z parseru).

```
CDonkey donkey = CDonkey("Aplikace pro deti", 800, 600, false );
frame = donkey.addFrame("skeleton");
frame->addImage( 50, 30, "./img/skeleton.png");
frame->addLabel( 430, 340, "chest");
frame->addLabel( 430, 370, "foot");
frame->addJointPane( 50, 30, 480, 550, 10,
0x5c8cdb, JOINTPANE_RANDOM_COLOR, "pane1" )
->addJointPair( 65, 130, 350, 210 )
->addJointPair( 206, 55, 350, 240  )
->addJointPair( 240, 115, 350, 270 )
```

Níže rozeberu nejdůležitější třídní API.

4.2.1.1 Konstruktor CDonkey

```
CDonkey(
const string & p_title,
Uint32 p_winWidth,
Uint32 p_winHeight,
bool p_fullscreen,
SFont * p_defaultFont,
SBackground * p_defaultBg = NULL,
SMusic * p_defaultBgMusic = NULL,
SCursor * p_defaultCursor = NULL,
SButtonSurface * p_defaultButSurface = NULL
);
```

Vytvoří jakýsi podklad aplikace. Konstruktoru můžeme předat rozměry, popřípadě volbu zobrazení okna na celé obrazovce. Ukazatele na struktury SBackground, SMusic, SCursor, SButtonSurface umožňují nastavení výchozích vlastností pro každý rám aplikace, od pozadí, podkladové hudby až po použitý font či obrázek kurzoru.

4.2.1.2 Metoda addFrame

```
CFrame * addFrame( const string & p_name );
```

Funkce se volá na objekt vytvořené instance CDonkey. Přidá do Donkey obrazovku, později odkazovatelnou pomocí proměnné v *p_name*. Záleží na pořadí přidávání rámů, jelikož se zobrazují postupně jak jsou definovány, není-li stanoveno jinak.

4.2.1.3 Metoda run

```
void run( void );
```

Po zavolání předá řízení hlavnímu běhovému vláknu a začne vykonávat nadefinované operace. Zobrazí okno na obrazovce a první definovaný rám.

4.2.1.4 Ostatní veřejné metody

Další veřejné metody třídy CDonkey slouží k nastavení vlastností aplikace jako jsou: titulek, výchozí pozadí, hudba, font, rozlišení obrazovky a jiné.

4.2.2 CFrame

Třída, která popisuje jednu obrazovku, neboli rám aplikace. Tvoří ji pozadí, podkladová hudba, použitý font a soubor viditelných komponent CControl (viz 4.2.3). Jedná se o celek sdružující nadefinované komponenty, které jsou posléze složeny v jednu obrazovku a vykresleny.

4.2.2.1 Konstruktor CFrame

```
CFrame(
const string & p_name,
SBackground * p_defaultBg,
SMusic * p_defaultBgMusic,
SFont * p_defaultFont,
SCursor * p_defaultCursor,
SButtonSurface * p_defaultButSurface
);
```

Konstruktor je parametry velmi podobný konstruktoru CDonkey s tím rozdílem, že zde už nelze uplatnit nepovinné parametry. Tyto jsou předány z hlavního rámce CDonkey. Konstruktor jako takový je volán výhradně přes metodu addFrame třídy CDonkey. Pro každý rám je nutno přidat sadu komponent zvlášť.

4.2.2.2 Metoda setNext

```
void setNext( const string & p_nextString );
```

Dovolí nastavení následujícího rámu, na který lze posléze odkazovat nejen pomocí jeho jména, ale i zástupným obecným symbolem (například *_nextOfPrevious*).

4.2.2.3 Metoda initFrame

```
void initFrame( SDL_Surface * p_window );
```

Metodu s předponou init má každý funkční prvek v mém objektovém návrhu. Jedná se o funkce, které inicializují, načítají a předpřipravují. V tomto kontextu metoda volá každou init metodu podřízenou a zajišťuje připravení multimediálního obsahu (např. vynucení načtení obrázků z disku) ještě předtím, než je okno zobrazeno na zobrazovacím zařízení. Do funkcí níže předává ukazatel na aktuální strukturu s detaily o vytvořeném okně (např. bitová hloubka, rozměry, ...), který je potřeba pro vykreslení některých komponent.

4.2.3 CControl

Důležitý prvek, který ve svých útrobách ukrývá základní, většinou virtuální metody. Každá nově nadefinovaná komponenta je nutně dědicem této třídy (viz. znázornění podřízených tříd 4.1).

```
CControl
├── CAnimation
├── CButton
├── CDrawPane
├── CImage
├── CImagePaint
├── CJointPane
├── CLabel
├── CMovePane
└── CPairs
```

Obrázek 4.1: Model dědičnosti tříd

Nejdůležitější vlastností je pozice, vyjádřená v obrazových bodech (pixelech), identifikační jméno (díky kterému je možno na komponentu odkazovat, popřípadě ji dodatečně ovládat nebo dodefinovat) a samotný typ komponenty vyjádřený číselně.

4.2.4 Konstruktor CControl

```
CControl(
Uint8 p_type,
Uint32 p_X,
```

```
Uint32 p_Y,
string p_variable
);
```

Spolu s komponenty souvisejí na jejich akci navázané události. Chování každé komponenty je dodefinovatelné a tak např. po klepnutí na komponentu tlačítko lze přejít na jiný rám, spustit animaci nebo vypnout celý program. Události jsou implementovány ve struktuře *SActions*, které se budu věnovat později v sekci 4.3.5.

4.2.5 Metoda getResult

```
virtual SDL_Surface * getResult( void );
```

Nejdůležitější virtuální metodou CControl je *getResult()*, která vrací strukturu *SDL_Surface* s výslednou grafikou komponenty.

4.3 Standartní komponenty a události

Sekce popisuje všude jinde standartně používané komponenty a události. Jejich možnosti jsou omezené na nutné minimum, ale nejdůležitější funkcionalitu mají zachovánu.

4.3.1 CImage

Jak již název napovídá, jedná se o komponentu vykreslující na obrazovku soubory s grafikou. Díky použití knihovny *SDL_image* je množina podporovaných formátů opravdu rozsáhlá. Soupis podporovaných typů grafiky, seřazených podle používanosti:

- BMP, GIF, JPEG, PNG
- LBM, PCX, PNM, TGA, TIFF, WEBP
- XCF, XPM, XV

4.3.2 CButton

Standartní klikací tlačítko se třemi základními polohami, které jsou znázorněny na obrázku 4.2. Pro jednotlivé stavy je samozřejmostí možnost definovat vlastní obrázky.

Definice obrázků poloh 2 a 3 je nepovinná. Pro uchování grafiky se stavy tlačítek existuje oddělená struktura s názvem *SButtonSurface*. Je možno ji vytvořit odděleně od tlačítka a propagovat například do rámu jako výchozí

- Normální (IdleSurface)
- Přikryté myší (HoverSurface)
- Stisknuté (ClickedSurface)

Obrázek 4.2: Příklad normálního, překrytého a stisknutého tlačítka

pro tento druh komponenty. Na události tlačítka je nutnost podpory navázání akcí. Možností je také vytvořit tlačítko neviditelné: stačí použít možnosti alfa-kanálu při tvorbě obrázku.

4.3.3 CLabel

Další standartní komponentou je CLabel neboli popisek. Popisky se mohou lišit použitým fontem, barvou a především obsahem. CLabel nepodporuje zobrazování textu pod určitým úhlem.

4.3.4 CAnimation

CAnimation je speciálně navržená komponenta pro zobrazování sekvencí po sobě jdoucích obrázků. Podporuje dva základní režimy přehrávání, a to spuštění v nekonečné smyčce nebo s předem daným počtem opakování. Třída má dvě základní metody *run()* a *stop()*, jejichž účel lze intuitivně odvodit. Grafické formáty lze načítat stejné jako pro komponentu CImage (viz 4.3.1), jelikož existuje společná funkce, která soubory načítá.

CAnimation vyžaduje načtení jednotlivých snímků, nikoliv hotovou animaci například v *.GIF*. Pro zjednodušení načítání je v parseru vytvořena možnost natáhnutí všech obrázků z daného adresáře seřazeně podle jména.

4.3.5 Událostní systém

Akce volané událostmi se dají navázat na kteroukoliv komponentu a dokonce i např. na načtení rámu. Důležitou vlastností je, že zaregistrovat lze více akcí na jednu událost. Akce jsou poté volány postupně podle pořadí, ve kterém byly definovány. Základní strukturou odpovědí na události je *SActions*.

4.3.5.1 Seznam podporovaných událostí

- OnClick
  klepnutí myší

- OnSuccess
  bezchybné vyřešení komponenty - ihned

- OnFault
  chybné vyřešení komponenty - ihned

- OnSuccessCalled
  přijde-li z jiného prvku požadavek na otestování komponenty a ta je vyřešena správně

- OnFaultCalled
  přijde-li z jiného prvku požadavek na otestování komponenty a ta je vyřešena chybně

- OnLoad
  rám se vykreslil

4.3.5.2 Seznam podporovaných akcí

- Navigate
  přechod na jiný rám

- Close
  ukončení aplikace

- Sound
  přehrání zvukového efektu pomocí jména, které je nadefinováno v Application kontextu

- Check
  dojde k otestování vyřešení komponenty a podle toho se zavolá navázaná akce OnSuccessCalled nebo OnFaultCalled

- Animate
  spustí animaci

- StopAnimate
  zastaví animaci

4.4 Specializované komponenty

Následující podsekce popisují možnosti komponent, speciálně navržených pro potřeby middlewaru pro vývoj aplikací pro děti. Mezi nimi můžeme nalézt „hry“ pro rozvoj motoriky, spojovačky, komponentu s názvem Pairs, díky níž lze vytvořit jednoduše a rychle interaktivní pexeso (složené z tlačítek) nebo například digitální omalovánky.

4.4.1 CDrawPane

Základním účelem *CDrawPane* je právě zlepšení motorických schopností dítěte. Je třeba relativně přesných pohybů myši pro pospojování jednotlivých bodů dle daného pořadí. Body jsou tzv. záchytné, tedy při vyjetí z dráhy nebo puštění tlačítka myši se automaticky čára vedoucí od posledního záchytného bodu ke kurzoru zruší a je třeba začít znova, nikoliv ale až od začátku. Tyto vlastnosti lze využít například pro obtahování tvarů písmenek (viz 4.3) nebo vytvoření jednoduchého bludiště (viz 8.8). Samozřejmostí je volitelná míra tolerance vzdálenosti od bodu, která už je považována za úspěšné spojení, barva čáry i bodu a jeho velikost.

CDrawPane vyžaduje předání parametru s odkazem na obrázek, který je posléze použit jako vodící podklad. Překročení černě vybarvené linie na podkladovém obrázku způsobí reset naposledy vytvořené čáry. Vyřešení komponenty dosáhneme při „dotažení“ čáry do posledního bodu v sekvenci.

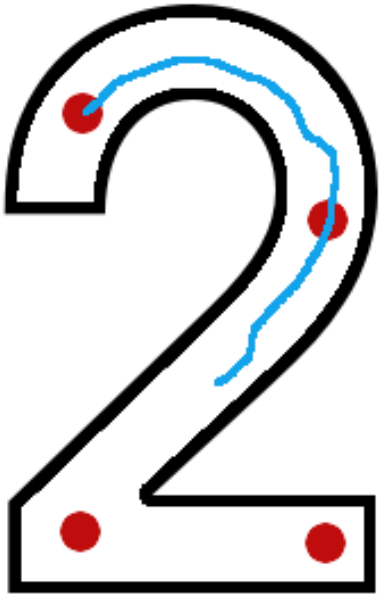
4.4.2 CJointPane

Joint je anglické slovo označující spoj nebo kloub. Z názvu vyplývá, že se jedná o komponentu, kde bude třeba něco spojovat. Hodí se např. k označování částí těla správným pojmem (viz 8.7). Pro starší žáky poslouží k procvičování anglických slovíček (spojováním slovíček s obrazovými výjevy, které popisují) nebo třeba procvičování matematiky (viz obrázek 4.4).

Počet spojovacích bodů na obrazovce je vždy nutně sudý (párují se vždy dva body k sobě). Lze spojit kterékoliv dva z nadefinovaných kloubů a při klepnutí na již spojený bod dojde k rozvázání vazby a lze jej znovu přepojit.

Komponenta jako celek je vyhodnocena na závěr spojování, tedy v okamžiku, kdy dojde k poslednímu navázání dvou zbývajících bodů. Vyhodnocena může být jako chybně nebo správně spojená a v závislosti na tomto výsledku je posléze vyvolána odpovídající událost, která zajistí spuštění návazných akcí.

Vykreslování spoje na obrazovku je zajištěno kvadratickou Bézierovou křivkou neboli plynulým spojem. Více se můžete dozvědět na webu A. Davidové a M. Víta [5], kde je celá problematika velmi přehledně vysvětlena. Lze měnit velikost a barvu bodu/linie.



Obrázek 4.3: CDrawPane komponenta

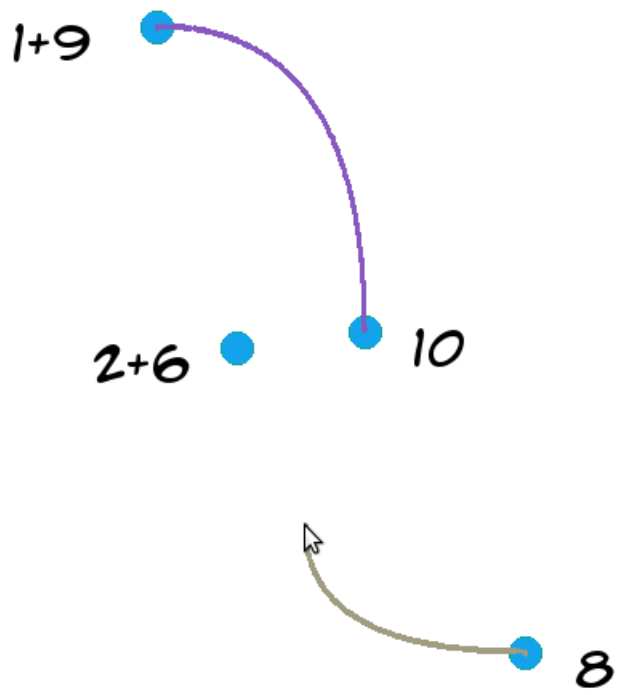
4.4.3 CPairs

Pairs neboli páry lze použít pro vytvoření pexesa (viz 4.5) nebo například hledání n rozdílů ve dvou obrázcích při použití dostatečně malých průsvitných tlačítek. V základu se jedná vlastně o dvě klikací komponenty CButton, které jsou na sobě vzájemně závislé. Při stisknutí jednoho prvku z páru je zapotřebí označit jeho dvojče. Pokud k tomu dojde, oba prvky se promění. Na druhou stranu, stisknu-li po sobě pár, který k sobě nepatří, oba se překlopí do základního stavu, ve kterém byly před označením. Celá komponenta je vyhodnocena jako vyřešená při správném „otočení“ všech karet. Stav „špatně vyřešeno“ zde nemůže nastat.

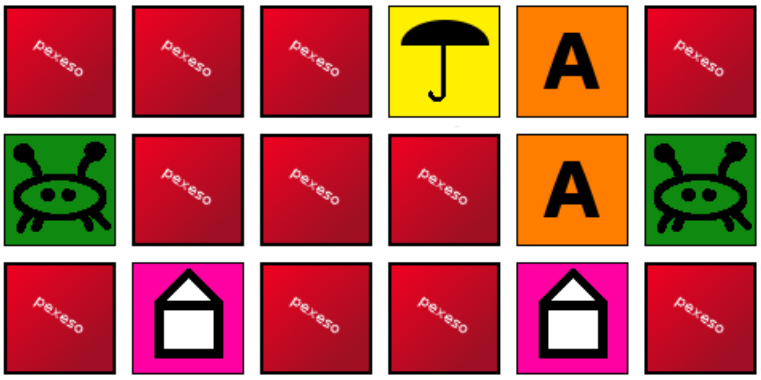
4.4.4 CImagePaint

CImagePaint nahrazuje omalovánky (viz 4.6). Spojité plochy ohraničené černou barvou lze vybarvit nadefinovanými plechovkami barev. Interně se pro vyplnění grafické plochy používá algoritmus založený na zásobníku *Flood fill*, neboli záplavové plnění.

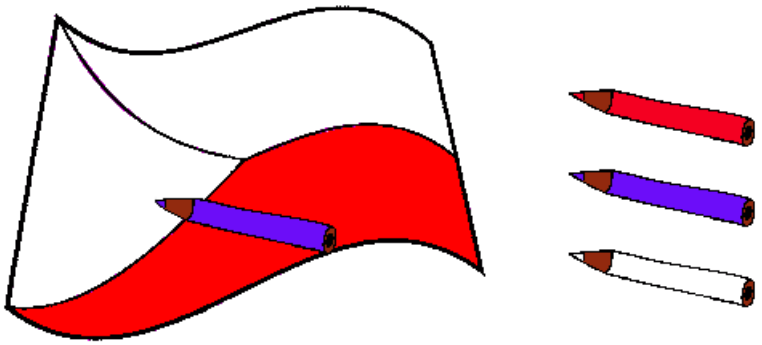
Lze nadefinovat body v podkladovém obrázku, které se budou kontrolovat pro vybarvení určitou barvou. Alfa kanál podkladového obrázku je zakázáno vybarvit, dá se tedy použít jako volná oblast, kam se nemá kreslit. Při kliknutí na obrázek, který reprezentuje barvu (lze nadefinovat vlastní), dojde k nahra-



Obrázek 4.4: CJointPane komponenta



Obrázek 4.5: CPairs komponenta



Obrázek 4.6: CImagePaint komponenta

zení kurzoru a jakmile je pravé tlačítko myši znovu stisknuto nad zamýšlenou oblastí, podklad se v rozsahu ohraničeném černou linií obarví.

Tuto funkčnost lze aplikovat například na procvičování barev na vlajkách různých států. Při každém novém aplikování barvy na podklad se vyhodnotí, zda je komponenta úspěšně vyřešena a následně se provedou nadefinované akce navázané na tuto událost. Stav „špatně vyřešeno“ zde opět nemůže nastat.

4.4.5 CMovePane

Na plátně (viz 4.7) *CMovePane* lze posunovat jeho jednotlivé fragmenty (obrázky) po ploše. Pokud jsou všechny jednotky umístěny na správných nadefinovaných místech, komponenta je vyřešena. Jednoduše lze nastavit toleranci vzdálenosti. Pro představu je například možné pomocí komponenty skládat jednotlivé části sněhuláka na sebe.

4.5 Konfigurační jazyk

Pro zajištění modifikovatelnosti vytvořených programů jsem vytvořil vlastní popisovací jazyk. Direktivy jsou psané v anglickém jazyce s formátováním UTF8 (kvůli přenositelnosti) a používají jednoduchou syntaxi. Přehlednost a intuitivnost pro mě byly hlavní cíle. Konfigurační soubor s doporučenou koncovkou *.dky* se dělí na dva základní kontexty direktiv: *Application* a *Frame*.

Jako oddělovač příkazů slouží znak dvojtečky „:“. Díky oddělovači je možné psát konfiguraci jednoho prvku do jediného řádku a tím šetřit místo a docílit tak zvýšené čitelnosti kódu. Popisovací jazyk je *case-insensitive*, tedy nezáleží na velikosti písmen v direktivách. Jednořádkové komentáře je možné psát pomocí znaku „#“, a to jak ze začátku řádku, tak i za direktivy.

Obrázek 4.7: CMovePaint komponenta

4.5.1 Neúsporný zápis jazyka

Kód lze psát velmi přehledně rozložený do více řádků. Všechny bílé znaky (mezery, nové řádky, tabulátory) jsou při zpracování parserem ořezány. Následuje ukázka nadefinování jednoduchého popisku pomocí komponenty *CLabel*.

```
Label
:Position("200","350")
:Text("Neusporny kod")
```

4.5.2 Úsporný zápis jazyka

Vlastnosti, patřící k určitému kontextu (Application, Frame, CLabel, CButton, CMovePane, ...), lze v jazyku psát s oddělovačem v řádku za sebe. Následující kód provede to samé jako ukázka výše.

```
Label:Position("200","350"):Text("Usporny kod")
```

4.5.3 Application kontext

Za pomoci direktivy *Application* (zapsané na začátku souboru) můžeme konfigurovat obecné chování a výchozí vlastnosti celého vytvářeného programu. Od textu, který se zobrazí v titulku okna, přes písmo, podkladovou hudbu až po rozlišení okna.

4.5.3.1 Vlastnosti

Následuje seznam možných nastavení a vlastností, konfigurovaných právě v kontextu *Application*.

- Title (nastavení titulku okna)
- Fullscreen (zobrazení na celou obrazovku)
- Resolution (rozlišení)
- BackgroundMusic (výchozí podkladová hudba)
- Font (výchozí font písma)
- Sound (definuje zvukové efekty pro pozdější použití)
- BackgroundColor (výchozí barva pozadí rámů)
- BackgroundImage (výchozí obrázek na pozadí rámů)
- Cursor (výchozí obrázek kursoru myši)

4.5.3.2 Ukázka použití

V této ukázce založíme aplikaci s rozlišením *1366x768* obrazových bodů, která se zobrazí v samostatném okně s názvem *Donkey BPM example 1*. Na pozadí bude hrát hudba s 60% zesílením a zároveň definujeme zvuk *click_sound* pro pozdější použití v událostech. Základní písmo bude černé *acmesa.TTF* velikosti 28 pixelů.

```
Application
:Title("Donkey BPM example 1")
:Fullscreen("no")
:Resolution("1366","768")
:BackgroundMusic("music/new_road_cars_soundtrack.mp3", "60")
:Font("fonts/acmesa.TTF","0x000000","28")
:Sound("sounds/cling.wav", "80", "click_sound")
```

4.5.4 Frame kontext

Vlastnosti zapsané do *Frame* kontextu ovlivňují chování jednotlivých obrazovek aplikace a zároveň direktiva slouží jako kontejner pro ostatní podřízené komponenty. Samozřejmostí je možnost změny pozadí, kurzoru nebo podkladové hudby. Tyto direktivy byly vyjmenovány již v kontextu *Application* (viz 4.5.3.1) a lze je aplikovat i zde.

4.5.4.1 Vlastnosti

Ve výpisu vlastností jsou dříve zmíněné vynechány.

- Name (název obrazovky pro odkazování)
- Next (hodnota „name“ další obrazovky)
- OnLoad (lze definovat akci, která se provede po načtení rámu)

4.5.4.2 Ukázka použití

V předchozím příkladu 4.5.3.2 jsme nadefinovali obecný rámec aplikace a nyní k němu přidáme rám. Kontext *Frame* píšeme do téhož konfiguračního souboru za konec kontejneru *Application*. Rám bude mít proměnné jméno *uvitani*, díky čemuž se na něj bude moci později odkazovat. Dalším rámem v pořadí nastavíme *pexeso*, které v době definice ještě nemusí být v souboru definováno. Změníme barvu pozadí a po načtení spustíme animaci s názvem *kralik* (bude definována později) a po osmi vteřinách přejdeme na obrazovku definovanou v direktivě *next*.

```
Frame
:Name("uvitani")
:Next("pexeso")
:OnLoad("animate","kralik")
:BackgroundColor("0x885A36")
:OnLoad("navigate","_nextFrame", "8")
```

Můžeme zde vidět hned dvě zajímavosti jazyka. První je, že například direktivu OnLoad lze opakovat a tím pádem navázat více akcí na jednu událost (v tomto případě animace a automatický přechod na další rám po 8 vteřinách). Druhou je zástupný symbol *_nextFrame*, který způsobí přechod na rám nadefinovaný ve vlastnosti „next“ této obrazovky.

4.5.5 Kontexty komponent

Každá vytvářená komponenta vlastní svůj kontext. Komponenty je nutné definovat pod určitým Frame. Každá má nadefinovanou minimálně pozici a další vlastnosti se liší podle typu. Nebudu se zde věnovat standartním komponentám (Label, Button, Image), ale raději detailněji rozepíši komponenty specializované. Vynechám přitom popis vlastností, jako jsou např. *Name*, *Position*, *Width*, *Height*, jejichž dopad je jednoduše odvoditelný. Popis účelu a funkčností těchto jednotek naleznete v sekci 4.4.

4.5.6 Kontext MovePane

4.5.6.1 Vlastnosti

- Precision (tolerance vzdálenosti od požadované pozice)
- AddImage (přidá hybatelný fragment)
- OnSuccess (lze zaregistrovat akce, které se provedou po dokončení)
- OnSuccessCalled (registrace akce po volání např. z tlačítka, pokud je hotovo)
- OnFaultCalled (registrace akce po volání např. z tlačítka, pokud není hotovo)

4.5.6.2 Ukázka použití

Ukázka vytvoří komponentu o velikosti celé obrazovky (aby bylo možné fragmenty posouvat po celé ploše) s tolerancí přesnosti 8 pixelů. Jednotlivé pohyblivé obrázky přidáme direktivou *AddImage*, kde první dva parametry jsou počáteční X a Y pozice a další dva značí pozici, kam je třeba obrázek přemístit. Tato komponenta bývá velmi často podložena pomocí *Image*, které alespoň nějakým způsobem znázorní zamýšlené pozice hybatelných obrázků. Po úspěšném vyřešení počkáme jednu vteřinu a poté přepneme na rám s názvem *success*.

```
MovePane
:Position("0","0")
:Width("1366")
:Height("768")
:Precision("8")
:AddImage("30","90","218","398", "img/krtek_a_sova_p1.jpg")
:AddImage("400","50","333","560", "img/krtek_a_sova_p2.jpg")
:AddImage("550","200","364","242", "img/krtek_a_sova_p3.jpg")
:OnSuccess("navigate","success","1")
```

4.5.7 Kontext JointPane

4.5.7.1 Vlastnosti

- PointColor (barva bodu pro spojování)
- LineColor (barva spojovací čáry, 0x654321 = náhodná barva)
- AddPair (přidá dvojici spojitelných bodů)
- OnSuccess (lze zaregistrovat akce, které se provedou po dokončení)

- OnFault (lze zaregistrovat akce, které se provedou při chybném pospojování)
- OnSuccessCalled (registrace akce po volání např. z tlačítka, pokud je hotovo)
- OnFaultCalled (registrace akce po volání např. z tlačítka, pokud není hotovo)

4.5.7.2 Ukázka použití

Následující kód vytvoří na pozici *50:130* blok o velikosti *480x550* pixelů. Barva bodů bude *0xA600A6* a barva spojovací čáry *0x654321* neboli náhodně vybraná. *AddPair* přidá dvojici bodů, které mají k sobě vztah. Po spojení všech se přejde buď na rám *success* při úspěchu nebo na *fault* při neúspěchu.

```
JointPane
:Position("50", "130")
:Width("480")
:Height("550")
:PointColor("0xA600A6")
:AddPair("65", "130", "350", "210")
:AddPair("206","55", "350", "240")
:AddPair( "240", "115", "350", "270" )
:AddPair( "160", "310", "350", "300" )
:AddPair( "202", "170", "350", "330" )
:AddPair( "150", "458", "350", "360" )
:LineColor("0x654321")
:OnSuccess("navigate","success")
:OnFault("navigate","fault")
```

4.5.8 Kontext DrawPane

4.5.8.1 Vlastnosti

- BaseImage (podkladový obrázek)
- PointColor (barva bodu pro spojování)
- LineColor (barva spojovací čáry, 0x654321 = náhodná barva)
- AddPoint (přidá průchodný bod - záleží na pořadí přidávání)
- OnSuccess (lze zaregistrovat akce, které se provedou po dokončení)
- OnSuccessCalled (registrace akce po volání např. z tlačítka, pokud je hotovo)

- OnFaultCalled (registrace akce po volání např. z tlačítka, pokud není hotovo)

4.5.8.2 Ukázka použití

Na pozici *100:60* bude vykreslen obrázek *bludiste.jpg* s černými okraji. Do něj jsme nadefinovali tři „body zájmu“. Jejich barva bude *0xA633A6* a při spojování bude použita barva *0xA600A6*. Po úspěšném vyřešení přejdeme po jedné vteřině na obrazovku s názvem *success*.

```
DrawPane
:Position("100", "60")
:BaseImage("img/bludiste.jpg")
:PointSize("10")
:PointColor("0xA633A6")
:LineColor("0xA600A6")
:AddPoint("79", "84")
:AddPoint("472","327")
:AddPoint("567", "581")
:OnSuccess("navigate", "success", "1")
```

4.5.9 Kontext ImagePaint

4.5.9.1 Vlastnosti

- Image (podkladový obrázek)
- ImageOffset (pozice obrázku od levého horního okraje komponenty)
- AddColorCan (přidá plechovku barvy)
- OnSuccess (lze zaregistrovat akce, které se provedou po dokončení)
- OnSuccessCalled (registrace akce po volání např. z tlačítka, pokud je hotovo)
- OnFaultCalled (registrace akce po volání např. z tlačítka, pokud není hotovo)

4.5.9.2 Ukázka použití

Nadefinujeme komponentu *ImagePaint* s názvem *vlajka_paint* na *X:Y* pozici *120:120* s rozměry bloku *600x300*. Podkladový obrázek, do kterého se bude kreslit, *czech-flag-wa.png* odsadíme od horního a levého okraje bloku o *50* pixelů. Tuto vlastnost definujeme z důvodu, že při vyjetí z rozměrů bloku s vybranou barvou dojde k resetu kurzoru. Pomocí *AddColorCan* přidáváme

jednotlivé barvičky na výchozí pozici X, Y, zastoupené daným obrázkem s definovanou barvou. Jako poslední parametry předáváme direktivě *AddColorCan* dvojice bodů v obrázku, které se mají kontrolovat na danou barvu.

```
ImagePaint
:Name("vlajka_paint")
:Position("120","120")
:Width("600")
:Height("300")
:ImageOffset("50","50")
:Image("img/painting/czech-flag-wa.png")
:AddColorCan("400","100","red.png", "0xFF0000", "173", "150")
:AddColorCan("400","150","blue.png", "0x0000FF", "42", "104")
:AddColorCan("400","200","white.png", "0xFFFFFF", "150", "50",
"100", "50")
```

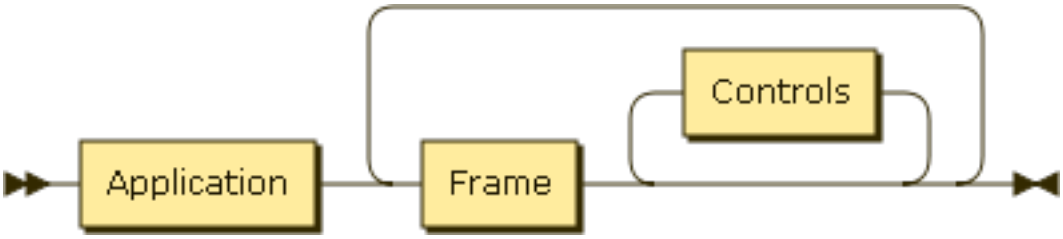
Do jedné barvy lze pomocí popisovacího jazyka nadefinovat i více kontrolních bodů:

```
# obecná definice
:AddColorCan("x","y","obrázek", "barva",
"p1X", "p1Y"[, "p2X", "p2Y"[,..]])

# kontrola jednoho bodu
:AddColorCan("400","200","white.png", "0xFFFFFF",
"150", "50")

# kontrola dvou bodů 150:50 a 100:50
:AddColorCan("400","200","white.png", "0xFFFFFF",
"150", "50", "100", "50")
```

4.6 Modul pro HTTP protokol

Komunikace s centrálním distribučním serverem probíhá po síti za pomoci přenosového protokolu HTTP. Operační systém Windows používá trochu odlišný způsob síťování než UNIX a tudíž bylo nutné sáhnout po další multiplatformní knihovně. *Boost* je neziskový projekt, který poskytuje variaci všemožných nadstaveb pro jazyk C++ a především dvě varianty knihoven, pro UNIX i Windows systémy. Osobně jsem využil funkcí pro práci s filesystémem a síťováním (modul asio [9]).

Třída CHTTP je definována nezávisle na *CDonkey*. Využívá možností TCP/IP a je uzpůsobena přímo pro stahování souborů s pomocí HTTP verze 1.1. Verze 1.1 na rozdíl od té předchozí 1.0 podporuje mimo jiné i virtuální



Obrázek 4.8: Gramatika - základ

hostitele, což je velmi důležitá vlastnost, neboť soubory aplikace je vždy nutno stahovat z kořene webového serveru.

Po připojení klienta na server je vyžadován soubor */donkey.idx*, který obsahuje cesty ke všem souborům, které je potřebné pro běh aplikace stáhnout. Díky funkcionalitě *Boostu* je možné připojit se jak na IP adresu stroje, tak na jeho doménové jméno, které je následně přeloženo.

4.7 Modul pro parser

Dalším nezávislým modulem je třída, která se stará o zpracování a interpretaci konfiguračních souborů v jazyku *Donkey*. *CDonkeyParser* jako parametr v základu přebírá pouze cestu k onomu souboru a prázdný ukazatel na strukturu *CDonkey*, kterou následně naplní daty.

Každý zpracovávaný kontext má nadefinované povinné a nepovinné parametry a v důsledku užití šablon v kódu je docíleno celkové přehlednosti a jednodušší editovatelnosti modulu.

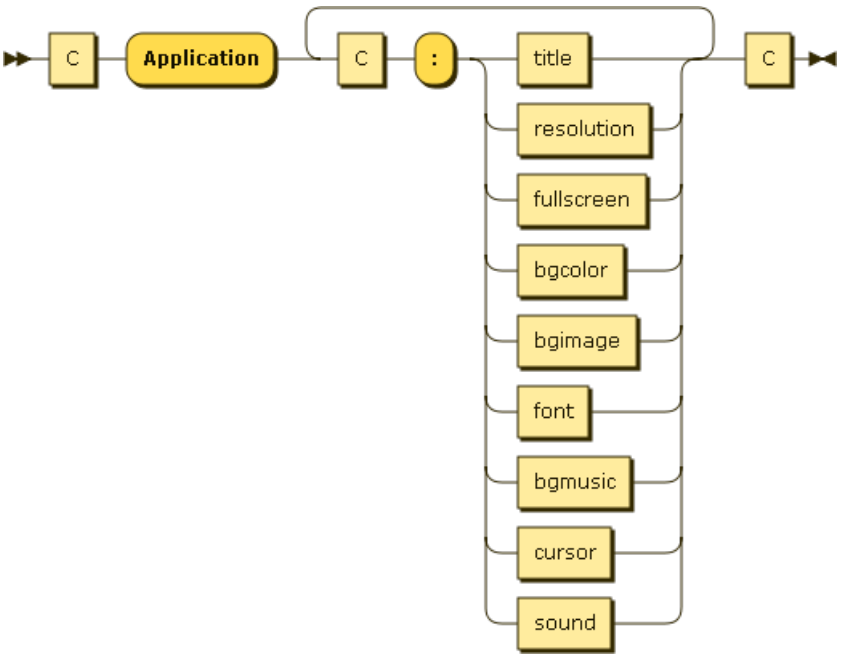
Parser využívá tzv. *agregator*, který se stará o detekci redundantních definic fontů, hudby, kurzorů, pozadí nebo tlačítek. Místo vytvoření nové instance pouze přesměruje odkaz na již vytvořenou třídu a tím šetří zdroje operačního systému a zrychluje celý běh aplikace.

4.7.1 Gramatika parseru

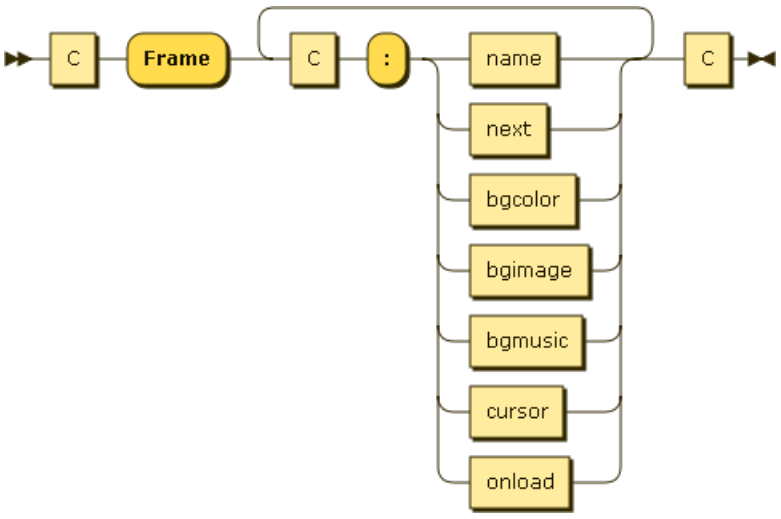
Gramatika v parseru je bezkontextová, typu LL(1). Následuje grafické znázornění gramatiky, vygenerované pomocí nástroje *Railroad Diagram Generator* [22]. Pod tímto textem jsou uvedeny pouze některé vybrané části gramatiky. Kompletní vygenerované diagramy celé gramatiky, jež generuje kompletní *Donkey* jazyk, naleznete na přiloženém CD.

4.7.2 Způsob realizace parseru

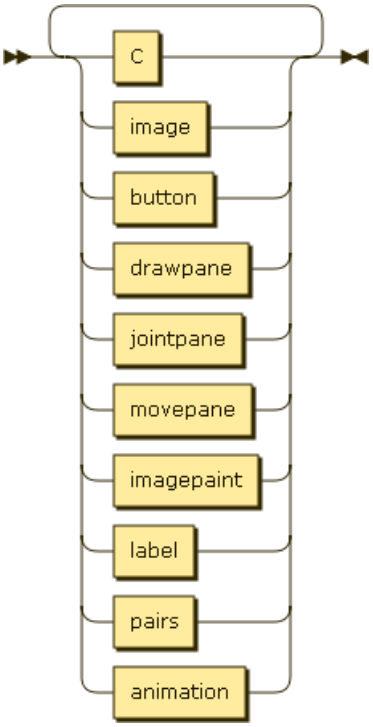
Pro realizaci parseru vlastního jazyka jsem se rozhodl nepoužít žádnou z nadstavbových knihoven usnadňujících zpracování jazyka. Hlavním důvodem pro mé rozhodnutí byla skutečnost, že gramatika samotného popisovacího jazyka



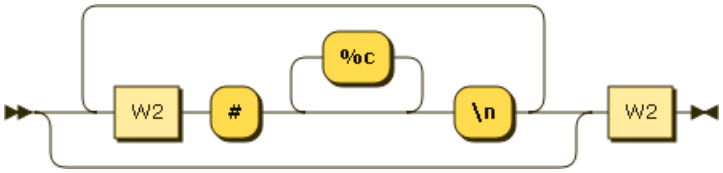
Obrázek 4.9: Gramatika - kontext Application



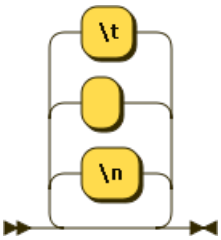
Obrázek 4.10: Gramatika - kontext Frame



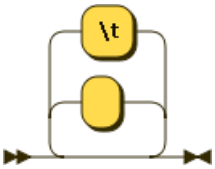
Obrázek 4.11: Gramatika - kontext Controls



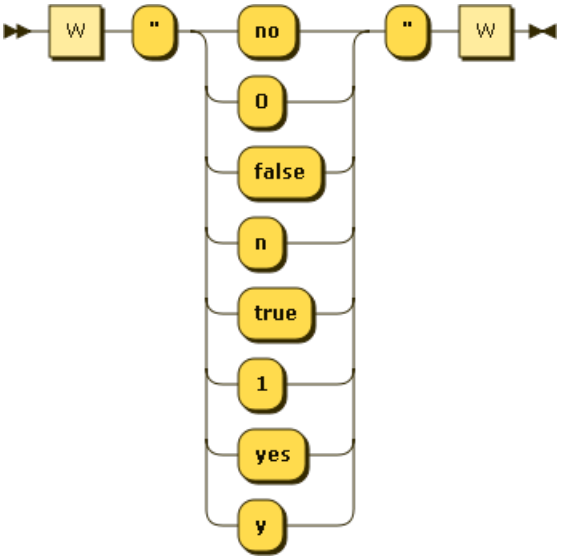
Obrázek 4.12: Gramatika - definice neterminálu C (bílé znaky a komentáře)



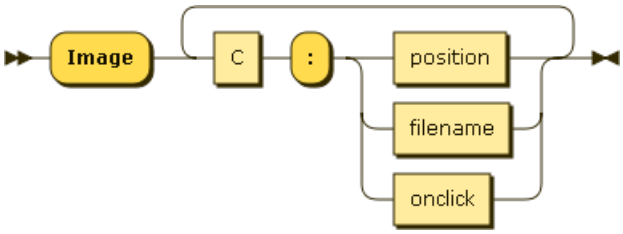
Obrázek 4.13: Gramatika - definice neterminálu W2 (jednořádkové bílé znaky)



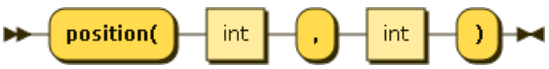
Obrázek 4.14: Gramatika - definice neterminálu W (všechny bílé znaky)



Obrázek 4.15: Gramatika - ukázka gramatiky načítání hodnoty - boolean



Obrázek 4.16: Gramatika - ukázka zástupce Controls - image



Obrázek 4.17: Gramatika - ukázka atributu - position

nebyla natolik rozsáhlá a tedy ani implementačně náročná, abych jí věnoval vyšší dávku úsilí. Pokud bych se rozhodl jít druhou cestou, použil bych například Bison pro C++ (který s sebou ovšem také nese nutnost učení se nové syntaxe zápisu).

4.7.2.1 Programová realizace

V kódu je nadefinováno pár základních načítacích funkcí, které se starají o validaci a uložení různých hodnot (číslo, uvozovkovaný text, pravdivostní hodnota, ...). Každý kontext (myšleno uzavřená logická oblast konfiguračního souboru - například Image, popisující obrázek) je realizován strukturou, do které se pomocí dříve zmíněných funkcí načítá obsah zadaný uživatelem. Tyto struktury jsou vkládány do dynamických polí a řazeny.

Při programování bylo myšleno na paměťové optimalizace a tudíž zde funguje koncept tzv. agregátoru. Agregátor zjišťuje, zda v systému již není nadefinován stejný objekt (obrázek, text se stejnými vlastnostmi, ...) a pokud ano, nevytváří tuto strukturu znovu, nýbrž pouze zavede symbolický odkaz na již vytvořený objekt. Tohoto se využívá zejména při definici fontů, obrázků nebo tlačítek.

Po úspěšném přečtení veškerých proměnných definujících daný kontext probíhá kontrola povinných argumentů. Chybí-li povinná vlastnost, program zobrazí chybovou hlášku s popisem problému. Tímto se docílí celkové konzistence vytvořené aplikace.

Oproti realizaci pomocí podpůrné knihovny má tento kód méně úsporný zápis, ale dle mého názoru je daleko lépe čitelnější. V každé fázi kódu je vidět, co bude program provádět. Případné doděláv̌ky či opravy je velmi jednoduché správně začlenit do kódu.

Kapitola 5

Testování

Následující kapitola se snaží zhodnotit celý middleware a vytvořené aplikace z hlediska funkčnosti, stability i uživatelské přívětivosti. Předpokládá se dodržování základních norem, jak pro úpravu kódu, tak např. pro korektní práci s pamětí a uživatelskou přívětivost.

5.1 Testy

Jednotlivé testy jsou rozděleny do následujících kategorií.

5.1.1 Testy práce s pamětí

Pro potřeby testování práce aplikace s pamětí bylo použito nástroje *valgrind* [27] a nadstavby pro grafické zobrazování *massif*. Objektově orientovaný jazyk C++ se od svých konkurentů (např. Java, C#) liší mimo jiné tím, že nemá vlastní garbage collector, neboli automatické uvolňování nepotřebné/dereferencované paměti. Tento úkon je tím pádem zcela na bedrech programátora aplikace.

Největším úskalím při testování aplikací vytvořených pomocí middlewaru je skutečnost, že samotné SDL (viz 3.1.3.1) obsahuje tzv. memleaky (neuvolněnou paměť) při konci programu. Po zkompilování nejjednoduššího možného C++ kódu obsahujícího základní funkcionalitu SDL nám nástroj *valgrind* zahlásí spoustu špatně uvolněné paměti. V mé implementaci jsem dbal na to, aby každý objekt vytvořený na haldě byl při zrušení řádně dealokován. Nástroj *massif* při běhu ukázkových aplikací nevykazoval žádné anomálie v používání paměti.

5.1.2 Testy multiplatformnosti

Aplikace byly úspěšně nasazeny na stroje s operačním systémem od společnosti Microsoft, jmenovitě: *Windows 7 Professional, Windows 7 Professional*

N, Windows 7 Home a Windows XP. Z linuxových distribucí to byly *Linux 17 Qiana, Ubuntu 12.04LTS, Ubuntu 14.04LTS, Debian 7 Wheezy LXDE*. Příprava systémů pro otestování korektního běhu je velmi časově náročná a tudíž zde uvedené platformy jsou značně omezené co do počtu.

Do budoucna by bylo vhodné rozšířit testovací platformy na linuxové stroje jiné rodiny než Debian, například *OpenSuse, Fedora, Arch nebo Gentoo*. Zdá se však, že nejdůležitější podmínkou pro běh aplikace vytvořené middlewarem je přítomnost podpůrných knihoven (SDL, Boost) na daném systému. Middleware byl vyvíjen v linuxovém prostředí, a proto se očekávaně při přenosu na Windows vyskytly nemalé problémy detailněji popsané níže.

5.1.2.1 MPEG-2 Audio Layer III

Po první úspěšné kompilaci zdrojového kódu a následném spuštění ukázkové aplikace na platformě Windows se vyskytl problém s přehráváním hudebních souborů ve formátu MP3 (MPEG-2 Audio Layer III). Průběh aplikace byl korektní, ale při pokusu zavřít aplikaci se na ploše objevilo výstražné okno s chybou. Tuto chybu se po dlouhém bádání podařilo nalézt v SDL knihovně *smpeg.dll*. Na linuxových strojích se však tato obtíž nevyskytla a proto jsem po několika neúspěšných pokusech odstranit chybu byl nucen dočasně zakázat podporu importu MP3 audio souborů do aplikace.

5.1.2.2 Kódování UTF-8

Další obtíže nastaly při přenosu konfiguračního souboru vytvořeného v prostředí linuxu, kde je jako nativní kódování textových souborů používáno *UTF-8*, do Windows. Pokud totiž uživatel edituje takto přenesený konfigurační soubor některým neinteligentním textovým editorem (např. notepad), který nezachová původní kódování, a uloží např. v *CP-1250*, aplikace při spuštění nahlásí nekonzistenci tohoto souboru. Je tedy důležité při jakékoli úpravě těchto informací používat výhradně kódování *UTF-8*.

5.1.3 Testy distribučního systému

Otestování funkčnosti síťových modulů a distribuce po lokální síti LAN proběhlo kompletně ve virtualizovaném prostředí *Virtualbox VM* [16]. Jako hlavní distribuční server byl vybrán systém *Debian 7 Wheezy* s nainstalovaným webovým serverem *HTTPD Apache 2* a předkonfigurovanou složkou s aplikací na distribuci. Rozdělování síťových adres měl na starosti samotný Virtualbox. Klienti (s nainstalovanými knihovnami pro běh), postupně různé systémy z řad Linuxu i Windows, neměly sebemenší problém při připojení, stažení aplikace a následném spuštění.

5.1.4 Uživatelské testy

Zřejmě nejobjektivnější hodnocení významnosti či užitečnosti middlewaru může poskytnout zpětná vazba od budoucích tvůrců multimediálních aplikací. Předal jsem proto vývojové prostředí pro otestování postupně pěti lidem, kteří měli pouze základní zkušenosti s programováním, aby zhodnotili obtížnost vytváření aplikací v middlewaru. Tímto bych jim také rád poděkoval za jejich čas.

5.1.4.1 Metodika

Úkolem pro testující bylo zhodnotit míru složitosti psaní aplikací, intuitivnost jazyka a možnosti middlewaru. Hodnotící se měli zaměřit především na následující body.

- Jazyk
- Dokumentace
- Výsledná aplikace

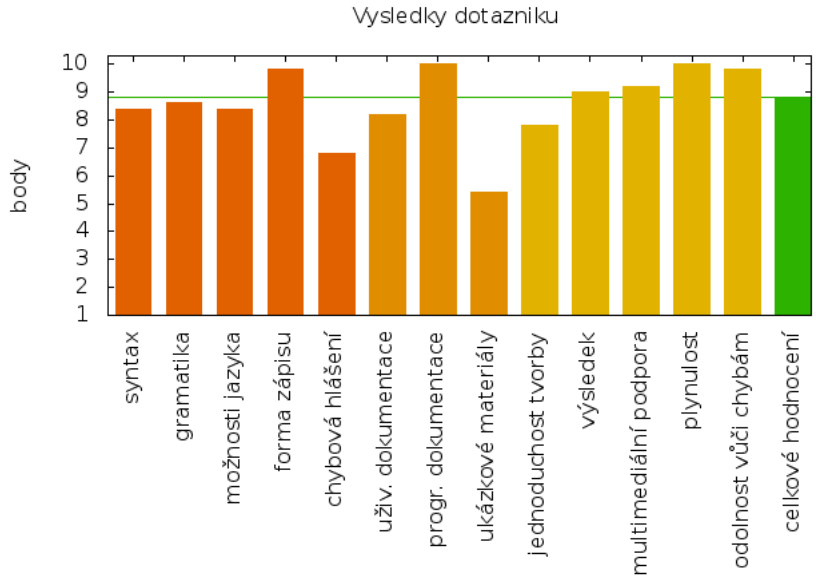
Detailní popis metodiky testování (dotazník) naleznete v příloze C.
Na obrázku 5.1 můžete vidět zprůměrované výsledky hodnocení jednotlivých kategorií uživatelských testů. Kategorie jsou barevně odděleny postupně: jazyk, dokumentace, výsledná aplikace a celkové hodnocení. Graf je vygenerován pomocí nástroje *Gnuplot* [8].

Jedna výtka směřovala na samotný jazyk, a to, že není použito pro konfiguraci některého z existujících značkovacích jazyků (např. standardů XML). Dalším nedostatkem, se kterým jsem již před začátkem testování počítal, bylo malé množství ukázkových konfiguračních souborů, kterými by se vývojář mohl rychle inspirovat.

Do budoucna, pokud by se middleware ujal, by se jistě našla širší komunita přispěvatelů, která by své „zdrojové kódy“ k aplikacím publikovala na veřejnosti. Všeobecně panovala spokojenost s intuitivností jazyka a očekávaným chováním aplikace. Možnosti jazyka (komponent) pokryly základní potřeby pro tvorbu jednodušších multimediálních aplikací. Výhledově by bylo velmi vhodné vytvořit IDE prostředí pro generování těchto konfiguračních souborů, neboť nyní při pozicování všech elementů do aplikace je často potřeba používat některého z grafických programů pro odečítání souřadnic.

5.2 Závěr testů

Nejvíce času vyhrazeného na opravování nalezených chyb nakonec zabralo přenesení middlewaru na druhý operační systém. Celkově však během testů middleware nevykazoval známky závažnějších chyb. Je třeba ovšem myslet na



Obrázek 5.1: Výsledky dotazníku z přílohy C

to, že samotná objemnost práce s sebou nese spousty chyb skrytých, které se budou postupně objevovat až při případném reálném provozu aplikace. Chyby odhalené během samotného testování byly úspěšně eliminovány a celý middleware je plně funkční.

Kapitola 6

Centrální distribuční server

Nasazení centrálního distribučního serveru má ulehčit práci např. učitelům v počítačových učebnách. Na hlavním stroji bude spuštěn HTTP webový server a připravena struktura souborů pro stažení a následné spuštění klientem. Server tímto bude umožňovat pohodlnou distribuci nových verzí aplikace.

6.1 Konfigurace serveru

Pro příklad jsem vybral operační systém Debian (velmi často používaný právě jako webserver) z rodiny Linuxových distribucí, a to ve stabilní verzi *7.4.0* architektury AMD64. Pro konfiguraci prostředí mi velmi pomohly výukové materiály pro předmět BI-AWD na FITu od Ing. Lukáše Bařinky [3] a web M. Dočekala [6] o konfiguraci HTTPD Apache.

6.1.1 HTTPD Apache

Při psaní tohoto dokumentu je webserver Apache ve verzi 2.4, ale zdroje Debianu stále používají stabilní verzi *2.2*. Stažení a instalace se provádí velmi jednoduše, jelikož je HTTPD Apache zařazen v oficiálních repozitářích Debianu.

6.1.1.1 Stažení a instalace

Následující krok je třeba provést pod oprávněním správce (root), tedy použijeme *sudo*.

```
sudo apt-get update
sudo apt-get install apache2
```

Webový server se automaticky po úspěšném provedení příkazu spustí a ve většině případů se také zařadí mezi *daemony*, spouštěné automaticky po startu operačního systému.

6.1.1.2 Nastavení naslouchání

Pomocí programu *netstat* můžeme zjistit, na kterých portech webový server poslouchá. Výchozí nastavení je port *80* na všech *interface*.

```
netstat -tlp --numeric-ports
```

V souboru */etc/apache2/ports.conf* můžeme dodefinovat IP adresy a porty, na kterých chceme požadavky obsluhovat. Chceme-li server nechat naslouchat na všech dostupných adresách s výchozím HTTP portem a povolit podporu virtuálních hostitelů, necháme v souboru následující direktivy:

```
NameVirtualHost *:80
Listen 80
```

6.1.1.3 Vytvoření virtuálního hostitele

Modul *CHTTP* middlewaru používá protokol HTTP ve verzi 1.1, tudíž podporuje jmenné (name-based) virtuální hostitele. Definice virtuálních hostů jsou standartně pro Debian umístěny v adresáři */etc/apache2/sites-available* v jednotlivých souborech. Nový záznam *donkey* provedeme např. zkopírováním a upravením souboru *default*:

```
cd /etc/apache2/sites-available
sudo cp default donkey
```

Následně soubor donkey editujeme např. editorem *nano*, samozřejmě stále s právy roota:

```
sudo nano donkey
```

Takto nějak by mohl vypadat výsledný konfigurační soubor virtuálního hostitele *donkey*.

```
<VirtualHost *:80>
  ServerAdmin webmaster@localhost
  ServerName www.donkey.edu
  DocumentRoot /var/www/donkey

  <Directory /var/www/donkey>
    AllowOverride None
    Order allow, deny
    Allow from all
  </Directory>

  ErrorLog /var/log/apache2/donkey_error.log
  LogLevel warn
</Virtualhost>
```

Pokud používáte program *nano*, klávesová zkratka *Ctrl+O* zajistí uložení a *Ctrl+X* následné zavření aplikace. Poslední kroky v nastavení webserveru jsou vytvoření hlavního adresáře projektu, načtení nové konfigurace a následný restart HTTPD Apache.

```
mkdir /var/www/donkey
sudo a2ensite donkey
sudo service apache2 restart
```

Nyní můžeme virtuálního hostitele otestovat například programem *nc* následovně:

```
nc localhost 80 << EOM
GET / HTTP/1.1
Host: www.donkey.edu

EOM
```

Pokud odpověď serveru je *200 OK*, vše je správně nastaveno.

6.2 Příprava adresářové struktury

Do adresáře nadefinovaného v direktivě *DocumentRoot* souboru */etc/apache2/sites-available/donkey*, v našem případě */var/www/donkey*, nakopírujeme všechny soubory naší nové aplikace (v ukázce používám testovací aplikaci example1).

```
/var/www/donkey/
├── example1/
│   ├── animations
│   ├── fonts
│   ├── img
│   ├── music
│   ├── sounds
│   └── example1.dky
└── donkey.idx
```

Následně vygenerujeme do kořene webového serveru klientem vyžadovaný indexový soubor *donkey.idx* např. tímto příkazem:

```
cd /var/www/donkey
find ./example1 -type f > donkey.idx
```

Soubor *donkey.idx* slouží klientovi k namapování adresářové struktury pro následné stahování. Soubor musí být umístěn v rootu HTTP serveru, jinak jej klient nenajde a zahlásí chybu. Pokud nevíme IP adresu serveru, můžeme ji zjistit z výpisu *ifconfig*, což se nám bude hodit pro účely testování v následující sekci.

```
ifconfig -a
```

6.3 Otestování klientem

Na klientském stroji s přeloženým běhovým prostředím je nutné nejdříve přidat DNS záznam pro www.donkey.edu do souboru */etc/hosts* v Linuxu a *%SystemRoot%\System32\drivers\etc\hosts* ve Windows. Do souboru přidáme řádek s *xxx* nahrazenou IP adresou *Donkey* webového serveru.

```
xxx www.donkey.edu
```

Následně spustíme klienta.

```
./donkey www.donkey.edu
```

Dojde ke stažení potřebných souborů ze serveru a spuštění aplikace.

Kapitola 7

Tvorba vlastní aplikace

Vytvořit vlastní aplikaci za pomoci middlewaru znamená z větší části napsat konfigurační soubor s použitím značkovacího jazyka a vytvořit multimediální obsah, který budeme chtít použít. Jazyk a celý proces tvorby byl navržen tak, aby bylo zapotřebí co nejmenšího úsilí a zkušeností ze strany tvůrce, při současném zachování volnosti kreativců.

7.1 Příprava

Ze začátku je třeba si připravit multimediální soubory, které v aplikaci hodláme používat. Obrázky, fonty, hudbu a všechny tyto materiály nakopírovat nejlépe do uzpůsobené adresářové struktury samotného projektu. Stromová struktura může vypadat například jako na obrázku 7.1.

```
projekt/
├── img
├── animations
├── fonts
├── music
├── sounds
└── projekt.dky
```

Obrázek 7.1: Doporučená struktura projektu

Přičemž soubor *projekt.dky* je konfigurační dokument zapsaný ve značkovacím jazyce. Po vytvoření struktury můžeme začít psát definici aplikace do tohoto souboru.

7.2 Základ aplikace

Základ aplikace se definuje pomocí direktivy *Application*, která musí být umístěna na začátku konfiguračního souboru.

```
Application
  :Title("Ukazkovy projekt")
  :Resolution("1366","768")
```

Application kontext poskytuje více nastavení (viz 4.5.3.1).

7.3 Definice obrazovek

Máme-li základ hotový, můžeme začít přidávat jednotlivé obrazovky. K tomu slouží direktiva *Frame*. Tu píšeme do stejného souboru jako *Application* a záleží nám na pořadí.

```
Application
  :Vlastnosti

Frame
  :Vlastnosti

Frame
  :Vlastnosti
```

Počet nadefinovaných obrazovek není middlewarem limitován - píšeme je opět do jednoho souboru pod sebe.

7.4 Triky pro testování

Pro rychlejší testování aplikace je vhodné do prvního načítaného rámu přidat na začátek direktivu *OnLoad* s akcí *navigate* a jako parametr uvést právě konfigurovaný rám. Po spuštění aplikace se rovnou úvodní obrazovka změní na požadovanou a není třeba procházet celou aplikací až ke kýženému rámu.

Kapitola 8

Ukázková aplikace

Pro potřeby testování i ukázku funkčnosti middlewaru jsem vytvořil prototyp aplikace. Skládá se z konfiguračního dokumentu a multimediálních souborů. Tento výtvor by měl sloužit jako předloha pro realizátory vlastních *Donkey* aplikací. Na CD disku v kapitole B jej naleznete pod názvem *example1*. Při jeho tvorbě jsem se snažil pokrýt veškeré dostupné funkce a vlastnosti middlewaru. Nápady na realizaci různých zadání obrazovek jsem čerpal také z knihy autorky K. Bäcker-braun, *Rozvoj inteligence u dětí od 3 do 6ti let* [2].

8.1 Požadavky

Nutností je přeložený zdrojový kód middlewaru do podoby běhového prostředí (binární soubor) podle operačního systému, na kterém bude aplikace spuštěna. Pro nastavení kompilátoru ve Visual Studiu je příhodné použít návod Setting up SDL in Visual Studio.NET [10].

8.2 Spuštění

Při spouštění nadefinované aplikace můžeme jít dvěmi cestami.

8.2.1 Spuštění z lokálního umístění

Jsou-li všechny potřebné soubory aplikace umístěny na lokálním disku a připraveno běhové prostředí ve formě aplikace, buď pomocí příkazového řádku nebo úpravou vlastností aplikace, zadáme jako první parametr cestu k hlavnímu konfiguračnímu souboru (má příponu *.dky*) a spustíme. Nezáleží přitom na typu adresy, můžeme zadat jak relativní tak absolutní cestu k souboru.

```
./donkey /home/donkey/projekt1/projekt.dky
Donkey.exe projekt1\projekt.dky
```

8.2.2 Spuštění ze vzdáleného umístění

Modul *CHTTP* nám umožňuje načítat aplikaci i ze vzdáleného serveru. Nutností je opět přeložené běhové prostředí a funkční síť mezi serverem a klientem. Místo cesty ke konfiguračnímu souboru zadáme buď IP adresu nebo *hostname* (doménový název) stroje, kde je správně připravena struktura souborů pro stažení. Po provedení tohoto kroku se na lokálním úložišti (tam, odkud byla aplikace spuštěna) zduplikuje stromová struktura nadefinovaná ve speciálním souboru *donkey.idx*, který je umístěn v kořenovém adresáři na serveru a middleware inicializuje běh programu.

Po skončení stromová struktura zůstává na lokálním úložišti a je tedy připravena k opětovnému spuštění, nyní již přes postup v předchozím kroku 8.2.1.

8.3 Průchod ukázkovou aplikací

Začátek průchodu aplikací je vyobrazen na obrázku 8.1. Jednoduchá uvítací obrazovka se automaticky po osmi vteřinách přepne na další v pořadí 8.2 díky direktivě *OnLoad* a akci *navigate* s volitelným druhým parametrem.

Následuje jednoduché procvičení matematických schopností dítěte na příkladu počítání kruhů (viz 8.3). Při chybné odpovědi dojde k odskočení na rám 8.4 na pět vteřin a zpět. Naopak při správném řešení se zobrazí obrazovka 8.5 a poté přeskočí na další rám v pořadí.

Díky vlastnostem navigate *_nextOfPrevious* a navigate *_previousFrame* lze tyto nadefinované obrazovky používat na více místech, aniž by byla zamezena globální funkčnost (tzv. odkazy z nich nejsou definovány napevno).

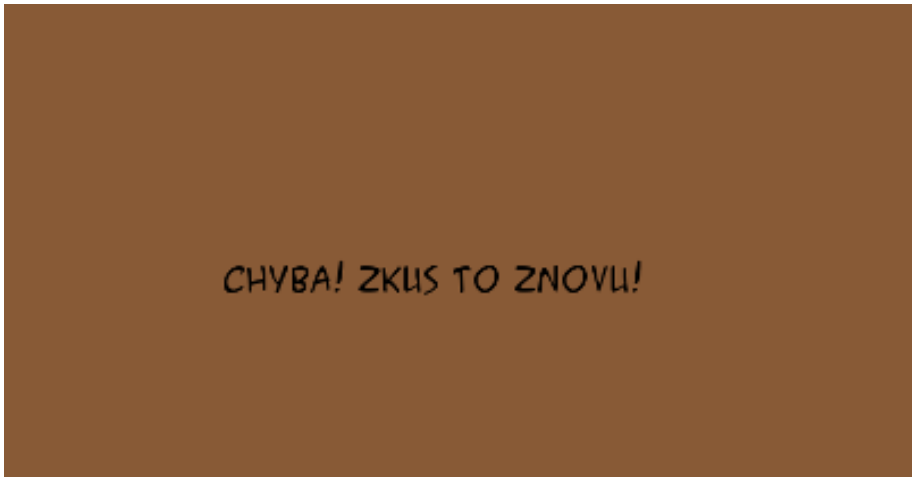
Čtvrtým hlavním rámem je kreslení vlajky 8.6, které funguje tak, že uživatel vybere barvu kliknutím, přesune kurzor s barvou nad zamýšlenou plochu a opětovným klikem dojde k obarvení celé oblasti. Po automatickém vyhodnocení se přejde na spojovačku lidské kostry s názvy jejích částí 8.7.

Jakmile úspěšně projdeme bludištěm 8.8 pospojováním bodů, čeká nás poslední hra 8.9, kde máme za úkol složit původní obrázek přetahováním jeho částí na správná místa. Pokud úspěšně vyřešíme i tuto, dostaneme se na závěrečnou obrazovku 8.10. Poté, co dojde ke kliknutí na obrázek hada, se ukončí celá aplikace.

Obrázek 8.1: Úvodní obrazovka ukázky (zdroje obrázků [15], [12])

Obrázek 8.2: Obrazovka 2 - pexeso (zdroj obrázku [17])

Obrázek 8.3: Obrazovka 3 - matematika (zdroj obrázku [14])

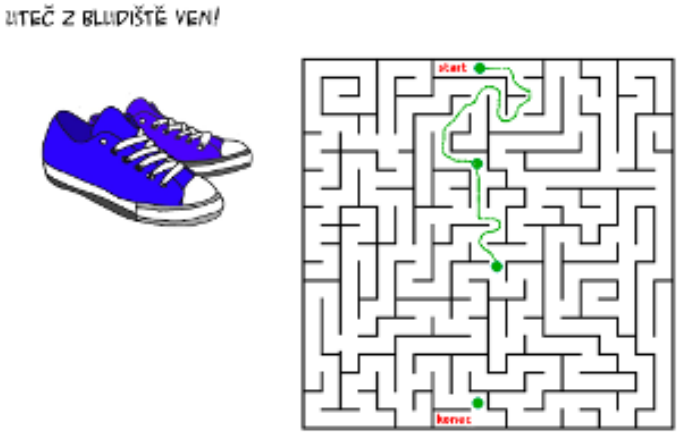


Obrázek 8.4: Rám oznamující chybnou odpověď

Obrázek 8.5: Rám oznamující správnou odpověď (zdroj obrázku [18])

Obrázek 8.6: Obrazovka 4 - kreslení

Obrázek 8.7: Obrazovka 5 - spojovačka (zdroj obrázku [19])



Obrázek 8.8: Obrazovka 6 - bludiště (zdroj obrázku [20])

Obrázek 8.9: Obrazovka 7 - skládání obrázku (zdroj obrázku [13])

Obrázek 8.10: Závěrečná obrazovka ukázky (zdroj obrázku [17])

Závěr

V této práci se podařilo realizovat middleware pro vývoj aplikací pro děti, který je spustitelný jak pod platformou Windows tak pod Linuxem. Pomocí programovacího jazyka C++ s nadstavbami se povedlo docílit dokonale portabilního software.

Součástí práce je vytvořený popisovací jazyk, který dovoluje používat různě tématicky zaměřené komponenty middlewaru a detailní konfigurovatelností umožňuje tvorbu aplikací na míru. Realizací jsou splněny všechny požadavky zadání a výsledný middleware je spustitelný. Lze vytvářet multimediální aplikace s minimem úsilí a zkušeností. Takto vytvořené programy podporují animace, zvuky, umožňují načítání obrázků a ovládání pomocí myši.

Nasazení výsledku mé práce je velmi vhodné do učebních prostor škol i domácností, pro výuku dětí předškolního a školního věku. Zároveň věřím, že má potenciál stát se významným zástupcem v oboru tvorby výukových aplikací. Neplánovaným vedlejším produktem je, že middleware lze nasadit dokonce jako např. prezentovací systém, který navíc podporuje více platforem. Z důvodu, že software nebyl prvoplánově pro tento účel vytvářen, je psaní prezentací časově náročnější, nýbrž možné.

V druhé části práce jsem se věnoval nastavení centrálního distribučního serveru, jeho instalaci a rozběhnutí ukázkové testovací aplikace. Řešení je určené především pro účely nasazení v učebních prostorách škol, které si nemohou dovolit kupovat drahé licence výukových programů. Hlavním přínosem je ušetření drahocenného času realizátorům a umožnění rychlé distribuce mezi žáky.

Samotný middleware je poměrně jednoduše rozšiřitelný o nové specializované komponenty a funkčnost. V budoucnu plánuji pracovat na rozvoji a věřím, že dokáže naplňovat podstatu, pro kterou byl vytvořen.

Literatura

[1] Aoki O.: *Debian reference*, 2013, [online][cit. 2014-09-22].
Dostupné z: `https://www.debian.org/doc/manuals/debian-reference/`

[2] Bäcker-braun K.: *Rozvoj inteligence u dětí od 3 do 6ti let*,
nakladatelství Grada Publishing a.s., 2014, I. vydání, ISBN 978-80-247-4798-9

[3] Bařinka, Lukáš.: *Přednášky a materiály k předmětu BI-AWD - Administrace webového serveru Apache 2*, 2014, [online][cit. 2014-10-02].
Dostupné z: `https://edux.fit.cvut.cz/courses/BI-AWD/start`

[4] cplusplus.com, *C++ Reference*, 2014, [online][cit. 2014-04-14].
Dostupné z: `http://cplusplus.com/reference/`

[5] Davidová, A. a Vít, M.: *Bézierova křivka*, 2013, [online][cit. 2014-05-18].
Dostupné z: `http://personal.tucna.net/cult/bezier.html`

[6] Dočekal, M.: *Správa linuxového serveru: Úvod do konfigurace Apache*, 2011, [online][cit. 2014-03-20].
Dostupné z: `http://linuxexpres.cz/praxe/sprava-linuxoveho-serveru-uvod-do-konfigurace-apache`

[7] Doxygen developers, *Doxygen documentation*, 2014, [online][cit. 2014-04-17].
Dostupné z: `http://stack.nl/~dimitri/doxygen/manual/index.html`

[8] Gnuplot developers and contributors, *Gnuplot - plotting engine*, 2014, [online][cit. 2014-11-09].
Dostupné z: `http://gnuplot.info/`

[9] Kohlhoff M. Ch.: *C++03 ASIO examples*, 2014, [online][cit. 2014-02-10].
Dostupné z: `http://boost.org/doc/libs/1_56_0/doc/html/boost_asio/examples/cpp03_examples.html`

[10] lazyfoo.net, *Setting up SDL in Visual Studio.NET 2010 Express*, 2012, [online][cit. 2014-03-28].
Dostupné z: `http://lazyfoo.net/SDL_tutorials/lesson01/windows/msvsnet2010e/index.php`

[11] Moreno C.: *C++ Vectors tutorial*, 2014, [online][cit. 2014-04-04].
Dostupné z: `http://mochima.com/tutorials/vectors.html`

[12] Nyström Jan-Eric: *Obrázky použité v ukázkové aplikaci (neupraveno) - animace koně*, 2014, [online][cit. 2014-10-25].
Dostupné z: `http://en.wikipedia.org/wiki/File:Animhorse.gif`

[13] Openclipart.org, Magnesus: *Obrázky použité v ukázkové aplikaci (neupraveno) - hedgehog*, 2014, [online][cit. 2014-10-25].
Dostupné z: `https://openclipart.org/detail/170650/small-hedgehog-by-magnesus-170650`

[14] Openclipart.org, Roland81: *Obrázky použité v ukázkové aplikaci (neupraveno) - bee*, 2014, [online][cit. 2014-10-25].
Dostupné z: `https://openclipart.org/detail/69817/cartoon-bee-by-roland81`

[15] Openclipart.org, vectorsme: *Obrázky použité v ukázkové aplikaci (neupraveno) - ant*, 2014, [online][cit. 2014-10-25].
Dostupné z: `https://openclipart.org/detail/181965/cartoon-ant-art-by-vectorsme-181965`

[16] Oracle developers, *Virtualbox dokumentace*, 2014, [online][cit. 2014-10-02].
Dostupné z: `https://www.virtualbox.org/wiki/Documentation`

[17] Pixabay.com, Nemo: *Obrázky použité v ukázkové aplikaci (neupraveno) - snake*, 2014, [online][cit. 2014-10-04].
Dostupné z: `http://pixabay.com/en/snake-serpent-green-reptile-303696/`

[18] Pixabay.com, OpenClips: *Obrázky použité v ukázkové aplikaci (neupraveno) - creature*, 2014, [online][cit. 2014-10-04].
Dostupné z: `http://pixabay.com/en/creature-cartoon-comic-happy-155759/`

[19] Pixabay.com, OpenClips: *Obrázky použité v ukázkové aplikaci (neupraveno) - skeleton*, 2014, [online][cit. 2014-10-04].
Dostupné z: `http://pixabay.com/en/skeleton-halloween-witch-corps-151170/`

[20] Pixabay.com, PublicDomainPictures: *Obrázky použité v ukázkové aplikaci (neupraveno) - shoes*, 2014, [online][cit. 2014-10-04].
Dostupné z: `http://pixabay.com/en/cartoon-drawn-blue-shoes-sneakers-163434/`

[21] Prus, V.: *Boost tutorial*, 2004, [online][cit. 2014-04-08].
Dostupné z: `http://www.boost.org/doc/libs/1_55_0/doc/html/program_options/tutorial.html`

[22] Rademacher G.: *Railroad Diagram Generator*, 2014, [online][cit. 2014-10-30].
Dostupné z: `http://www.bottlecaps.de/rr/ui`

[23] RFC 2616, *Hypertext Transfer Protocol – HTTP/1.1*, 1999, [online][cit. 2014-02-15].
Dostupné z: `http://tools.ietf.org/html/rfc2616`

[24] Smith, S.: *SDL Wiki*, 2014, [online][cit. 2014-04-11].
Dostupné z: `http://wiki.libsdl.org/FrontPage`

[25] The Apache Software Foundation, *Apache HTTP Server Version 2.4 Documentation*, 2014, [online][cit. 2014-03-02].
Dostupné z: `http://httpd.apache.org/`

[26] Turek, M.: *SDL: Hry nejen pro Linux*, 2005, [online][cit. 2014-04-18].
Dostupné z: `http://www.root.cz/serialy/sdl-hry-nejen-pro-linux/`

[27] Valgrind developers, *Valgrind uživatelská dokumentace*, 2014, [online][cit. 2014-10-02].
Dostupné z: `http://valgrind.org/docs/manual/manual.html`

Licence děl použitých v ukázkové aplikaci

Multimediální obsah použitý v ukázkové aplikaci může podléhat některé z následujících licencí

Creative Commons Attribution 4.0 International

http://creativecommons.org/licenses/by/4.0/legalcode

Creative Commons Public Domain

http://creativecommons.org/publicdomain/zero/1.0/deed.en

Příloha **A**

Seznam použitých zkratek

API Application Programming Interface

GUI Grafical User Interface

HTTP HyperText Transfer Protocol

IDE Integrated Developing Environment

IP Internet Protocol

JVM Java Virtual Machine

LAN Local Area Network

LTS Long Term Support

LXDE Lightweight X11 Desktop Environment

MP3 Moving Picture Experts Group Layer 3

MVC Model View Controler

OS Operační systém

UTF-8 UCS Transformation Format

Příloha B

Obsah přiloženého CD

```
/
├── bin/ ..... Složka s přeloženým middlewarem
│   ├── windows/
│   │   ├── Donkey.exe ..... Windows 7 middleware
│   │   └── *.dll ..... Knihovny nutné k běhu
│   └── linux/
│       └── donkey ..... Linux debian x86_64 middleware
├── docs/ ..... Dokumentace
│   ├── html/ ..... Doxygen dokumentace
│   ├── bp/ ..... PDF tohoto dokumentu
│   └── donkey/ ..... Příručka popisovacího jazyka + gramatika
├── example1/ ..... Ukázková aplikace
│   ├── animations/
│   ├── example1.dky ..... Konfigurační soubor
│   ├── fonts/
│   ├── img/
│   ├── LICENSE ..... Odkazy na autorská díla v ukázce
│   ├── music/
│   ├── README. ..... Jak spustit aplikaci
│   └── sounds/
├── INSTALL ..... Příručka pro překlad zdrojových souborů
├── Makefile ..... Makefile pro překlad na linuxu
├── README ..... Hlavní rozcestník CD
├── *.cpp. ..... Zdrojové soubory middlewaru
└── *.h. ..... Hlavičkové soubory middlewaru
```

Příloha C

Dotazník - testování middlewaru

Vytvořte prosím krátkou testovací aplikaci pomocí jazyka Donkey a přiložené dokumentace a zhodnoťte celý proces. V hodnocení následujících aspektů používejte, prosím, stupnici bodů od 1 do 10, přičemž 10 je nejvyšší spokojenost.

Hodnocení jazyka Donkey

Syntaxe - srozumitelnost značek

Zda anglické popisy a názvy jednotlivých sekcí/vlastností dávají smysl v kontextu vytváření aplikací.
1 2 3 4 5 6 7 8 9 10

Gramatika

Zda v gramatice jazyka nepostrádáte některou funkčnost.
1 2 3 4 5 6 7 8 9 10

Kreativní možnosti

Jednoduchost realizace vymyšlených nápadů (obrazovek a komponent) v jazyce Donkey.
1 2 3 4 5 6 7 8 9 10

Forma zápisu

Možnosti strukturalizace jednotlivých funkčních prvků v jazyce.
1 2 3 4 5 6 7 8 9 10

Chybová hlášení

Užitečnost informací z chybových hlášení při porušení pravidel jazyka.
1 2 3 4 5 6 7 8 9 10

Dokumentace

Uživatelská dokumentace k jazyku

1 2 3 4 5 6 7 8 9 10

Programátorská dokumentace

1 2 3 4 5 6 7 8 9 10

Množství ukázkových materiálů

1 2 3 4 5 6 7 8 9 10

Hodnocení výsledné aplikace

Jednoduchost tvorby

1 2 3 4 5 6 7 8 9 10

Shoda očekávání a výsledku

1 2 3 4 5 6 7 8 9 10

Podpora multimediálních typů

1 2 3 4 5 6 7 8 9 10

Plynulost aplikace

1 2 3 4 5 6 7 8 9 10

Odolnost aplikace vůči chybám a chybným vstupům

1 2 3 4 5 6 7 8 9 10

Celkové hodnocení

Celkové hodnocení práce jako celku vyjádřené body.
1 2 3 4 5 6 7 8 9 10

Doplňující poznámky/nápady

Děkuji.