VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

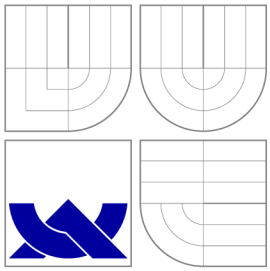
GUI KNIHOVNA PRO HRY NA PLATFORMĚ SDL2

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

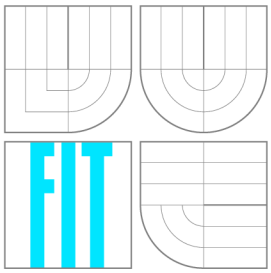
AUTOR PRÁCE
AUTHOR

JAN STANĚK

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

GUI KNIHOVNA PRO HRY NA PLATFORMĚ SDL2

GUI LIBRARY ON SDL2 PLATFORM DESIGNED FOR GAMES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JAN STANĚK

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. DRAHOSLAV ZÁŇ

BRNO 2015

Abstrakt

Tato práce se zabývá grafickými uživatelskými rozhraními v počítačových hrách jako prostředky pro komunikaci mezi uživatelem a aplikací. Dále se věnuje návrhu a implementaci obecného uživatelského rozhraní postaveného na platformě SDL2 a rozhraní OpenGL.

Abstract

This thesis examines the graphical user interfaces in computer games as a medium for communication between the user and the application. Then it describes the design and implementation of generic user interface using the SDL2 platform and the OpenGL specification.

Klíčová slova

GUI, SDL2, OpenGL, hry

Keywords

GUI, SDL2, OpenGL, games

Citace

Jan Staněk: GUI knihovna pro hry na platformě SDL2, bakalářská práce, Brno, FIT VUT v Brně, 2015

GUI knihovna pro hry na platformě SDL2

Prohlášení

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

. . . . . .
Jan Staněk
19. května 2015

Poděkování

Děkuji svému vedoucímu Ing. Záňovi za trpělivost a obětavost při konzultacích textové stránky práce. Dále děkuji všem, kteří mě při tvorbě práce podporovali a přiměli mě ji dokončit.

Obsah

1 Úvod **2**

2 Současnost uživatelských rozhraní **3**
2.1 Vývoj a vlastnosti uživatelských rozhraní 3
2.2 Věstavitelné knihovny pro tvorbu uživatelských rozhraní 4

3 Návrh knihovny **10**
3.1 Základní principy 10
3.2 Obecná struktura knihovny 11
3.3 Základní sada funkčních prvků 16

4 Implementace **19**
4.1 Popis výsledné knihovny 19
4.2 Prvky rozhraní 20
4.3 Vykreslování 21
4.4 Komunikace se zbytkem aplikace 24

5 Závěr **25**

Kapitola 1

Úvod

Počítačové hry se vyskytují prakticky v celé historii vývoje počítačů a ani v současnosti se tato skutečnost příliš nezměnila. S postupujícím vývojem hardwaru se objevují hry čím dál tím komplexnější, realističtější a náročnější na výpočetní výkon. S rostoucí komplexností také vzniká poptávka po znovupoužitelných řešeních některých oblastí tvorby her, které umožňují věnovat méně času implementaci podobných řešení a více času tvorbě originálního obsahu.

Jednou z těchto oblastí je uživatelské rozhraní, které pro hráče představuje hlavní způsob komunikace s hrou. Různé herní platformy kladou na tyto grafická uživatelská rozhraní různé požadavky a rozličná omezení. V důsledku těchto omezení se v uživatelských rozhraních her pro danou platformu objevují různé ustálené „zvyklosti“, tvořené obecnými prvky rozhraní, které nezávisle na typu hry plní podobné účely. Tento fakt jen zvyšuje poptávku po obecných knihovnách poskytujících implementaci těchto prvků.

Tato práce si klade za cíl vytvořit právě takovou obecnou knihovnu, poskytující implementaci obecné logiky prvků uživatelského rozhraní v takové podobě, aby bylo možné tyto prvky snadno použít v libovolné větší aplikaci či hře. Použitím moderních technologií, přenositelných na různé platformy včetně těch mobilních lze tohoto cíle dosáhnout, nebo se k němu alespoň přiblížit.

Následující kapitola této publikace se zabývá přehledem vlastností uživatelských rozhraní na různých platformách, spolu s typickými problémy těchto rozhraní a jejich běžnými řešeními. Dále jsou pak popsány některé existující nástroje a knihovny pro tvorbu uživatelských rozhraní, společně se zhodnocením jejich použitelnosti pro tvorbu rozhraní na běžném PC a mobilních zařízeních.

Třetí kapitola se věnuje návrhu. Jsou zde rozebrány základní principy návrhu a poté rozebrány jednotlivé oblasti pokryté navrženou knihovnou, společně s popisem relevantních problémů a jejich navrženým řešením. Ve další části této kapitoly je pak prezentována navržená struktura tříd a jejich vzájemné vztahy, společně s popisem navržených prvků rozhraní a jejich zamýšleným použitím.

Čtvrtá kapitola se zabývá implementací. Je zde krátce rozebrána použitá verze jazyka C++ a způsob řešení problémů vzniklých některými jeho omezeními na mobilních platformách. Dále jsou zde rozebrány algoritmy a technologie použité pro implementaci jednotlivých poskytnutých tříd.

Poslední, závěrečná kapitola shrnuje dosažené výsledky. Je zde uveden stav výsledné knihovny a způsob získání jejích zdrojových kódů. Také je zde výsledná knihovna srovnána s již existujícími řešeními a nastíněno možné budoucí pokračování práce.

Kapitola 2

Současnost uživatelských rozhraní

Tato kapitola se zabývá přehledem vlastností uživatelských rozhraní na různých zařízeních a v různých typech her. Jsou zde stručně popsány typické problémy, se kterými je třeba počítat při tvorbě grafických uživatelských rozhraní pro odlišné typy zařízení a některá jejich běžná řešení. Tato řešení jsou také demonstrována na příkladech se současné herní scény.

Následuje rozbor a srovnání čtyř dostupných věstavitelných knihoven pro tvorbu uživatelských rozhraní. Tyto knihovny jsou srovnávány s ohledem na jejich použitelnost při tvorbě her pro stolní počítače a mobilní zařízení.

2.1 Vývoj a vlastnosti uživatelských rozhraní

Způsob komunikace mezi počítačem a jeho uživatelem měl po dlouhou dobu relativně ustálenou podobu. Rozhraní aplikace bylo představováno jedním či více okny prezentujícími uživatelem požadovaný obsah. Kromě samotného obsahu tato okna poskytovala i různé ovládací prvky, například tlačítka pro zavření či skrytí daného okna.

Tento model vychází z představy, ve které obrazovka počítače představuje jakýsi virtuální pracovní stůl, na kterém uživatel manipuluje s jednotlivými dokumenty. Byl natolik zažitý, že pronikl i do situací, kdy metafora s pracovním stolem nebyla zcela odpovídající. Jako příklad lze uvést počítačovou RPG hru *Neverwinter Nights 2*, kde byla většina detailních dat o hráčově postavě prezentována právě pomocí plovoucích oken. Rozhraní obsahovalo i různé přehledové komponenty, takže při běžné hře nebylo nutné okna běžně používat, ale v případě detailnější správy charakteru byla herní scéna těmito okny velmi rychle překryta. Srovnání s nepříliš uspořádaným pracovním stolem se pak již nabízelo velmi zřetelně.

S příchodem moderních dotykových zařízení bylo nutné začít tento model přehodnocovat. Tyto zařízení mají mnohem větší rozptyl rozlišení displeje, takže je třeba dynamicky přizpůsobovat chování rozhraní různým velikostem displeje, a to dokonce i za běhu aplikace, například při natočení zařízení do svislé nebo vodorovné polohy. Dotykové ovládání také nabízí možnost ovládat rozhraní pomocí gest, která jsou rychlejší než hledání konkrétního tlačítka.

Vzhledem k předpokládané malé velikosti displeje na mobilních zařízeních se začal rozšiřovat model jedné aplikace (jednoho okna) na celou plochu zařízení a skrývání ostatních běžících aplikací na pozadí. Tyto celoplošná „okna“ také většinou nepoužívají tradiční ovládací prvky známé ze stolních počítačů, ale přizpůsobují se zařízením, na kterých běží. Běžná je snaha o co největší využití dostupných hardwarových ovládacích prvků, které poté není třeba replikovat softwarově a jejich nepřítomnost také šetří místo na displeji.

Uživatelská rozhraní ve hrách

Co se herních uživatelských rozhraní týče, ty byly a jsou přizpůsobovány zařízením, na kterých mají fungovat. Zatímco v případě her cílených především na stolní počítače do rozhraní pronikají prvky z typického „desktopu“ (viz příklad výše), u her určených i pro herní konzole je ovládání a rozhraní podřízeno možnostem této konzole.

Konzolové rozhraní se vyznačuje mimo jiné optimalizací na absenci myši či jiného ukazovacího zařízení. Také bývají časté poměrně velké a dobře čitelné popisy voleb – počítá se se hrou na televizní obrazovce ve větší vzdálenosti, než je průměrná vzdálenost od počítačové monitoru. Menší množství vstupních prvků na ovladači bývá řešeno zavedením několika různých módů hry. U her z pohledu první osoby může jít například o přepínání mezi běžným herním módem a nabídkovým módem. V módu běžné hry slouží ovladač pro pohyb a interakci s herním světem a grafické rozhraní má zejména informační funkci. V nabídkovém módu (může jít například o rozhovor s jinou herní postavou nebo o procházení předmětů v inventáři) slouží ovladač primárně vybírání z nabídnutých možností a neovládá herní svět přímo.

V případě konverze takové hry pro stolní počítač je pak ovládání mapováno na klávesy na klávesnici a tlačítka myši, není-li k dispozici konzolový ovladač. Grafické rozhraní ale zůstává prakticky nezměněno. Protože ale kombinace myši a klávesnice nabízí více možností než konzolový ovladač, nebývá mapování ovládání vždy přijato s nadšením. V případě populárních titulů se pak mohou vyskytnout neoficiální přepracovaní rozhraní, která dokážou využít možností nabízených výše zmíněnou kombinací. Příkladem může být hra *TES V: Skyrim* a její velmi populární modifikace *SkyUI*, která přidává mnoho nových možnosti oproti konzolové verzi rozhraní.

V případě mobilních zařízení je situace podobná jako v případě herních konzolí. Uživatelské rozhraní je přizpůsobováno specifickým vlastnostem a omezením cílového zařízení. Typické je ovládání pomocí gest a senzorů v zařízení, například simulace otáčení volantu nakláněním zařízení. Pokud už je součástí rozhraní nějaké tlačítko, je většinou relativně větší než na stolním počítači, a ideálně umístěno tak, aby co nejméně na displeji překáželo. Těmito zásadami se řídí i konverze her z jiných zřízení, i když se v tomto případě často volí cesta nejmenšího odporu.

Příkladem konverze starší hry pro dotykové zařízení může být aplikace *My Boy! Free*, což je emulátor přenosné herní konzole *Game Boy Advance* pro zařízení s operačním systémem Android. V tomto případě nejde o úpravu rozhraní samotné hry, ale o převedení ovládacích prvků výše jmenované konzole na dotykový displej. Ve výchozím nastavení aplikace vykresluje aproximaci ovládacích tlačítek konzole přes samotnou hru. Protože jde pouze o obrysové modely, příliš při hře nepřekážejí a lze tedy takto pohodlně hrát starší hru na novém zařízení s použitím známého ovládání.

2.2 Věstavitelné knihovny pro tvorbu uživatelských rozhraní

Pro tvorbu uživatelských rozhraní existuje množství volně dostupných knihoven a frameworků. Většina z nich je ovšem zaměřena na tvorbu rozhraní pro tradiční desktopové aplikace. Tyto knihovny zapouzdřují komunikaci se správcem oken operačního systému a poskytují prostředky pro tvorbu tradičních oken a jejich ovládacích prvků. Pro tvorbu specifických rozhraní, například pro trojrozměrné počítačové hry nebo mobilní dotykové aplikace, nejsou zcela vhodným nástrojem.

Oproti tomu existují i takzvané věstavitelné knihovny, které, jak již označení napovídá,

jsou navrhovány pro zabudování do větší aplikace. Ve většině případů vůbec nekomunikují se správcem oken operačního systému – tuto činnost přenechávají na aplikaci samotné. Místo toho zapouzdřují vykreslování ovládacích prvků pomocí nějaké obecné grafické vrstvy. Jako taková vrstva může sloužit třeba knihovna *SDL* nebo nějaká implementace specifikace *OpenGL*. Věstavitelné knihovny také zpravidla poskytují prostředky pro zachytávání událostí, týkajících se vykresleného rozhraní a mechanismus pro zpracovávání a reagování na tyto události.

Hlavní předností věstavitelných knihoven je právě jejich práce nad jinou vykreslovací vrstvou. Díky tomu je možné použít poskytované ovládací prvky již v existují grafické scéně – například pro vykreslování ukazatelů a jiných informačních prvků přes trojrozměrnou scénu herního světa, která již je plně v režii dané aplikace. Tvůrce takové aplikace pak nemusí trávit čas implementací například logiky a vykreslování stisku tlačítka, ale může se plně soustředit na herní svět a herní mechanismy.

QML

QML neboli *Qt Modelling Language* je deklarativní jazyk a knihovna prvků pro tvorbu uživatelských rozhraní[2]. Je vyvíjen jako součást frameworku Qt, konkrétně jako jeho modul *Qt Quick*. Je používán zejména pro tvorbu jiných než tradičních desktopových aplikací (tuto oblast pokrývají jiné části frameworku Qt), se zřetelným zaměřením na tvorbu rozhraní pro dotyková zařízení. Syntakticky připomíná jazyk JavaScript, a na tento jazyk je také úzce vázán – celou aplikaci lze napsat pouze pomocí QML a JavaScriptu. Implementován je ovšem v jazyce C++, a tímto jazykem jej také lze rozšiřovat.

Tvorba rozhraní probíhá pomocí deklarace vlastností jednotlivých prvků, mezi které patří umístění na ploše aplikace nebo chování při aktivaci prvku, a provázáním těchto prvků. Pro definici rozmístění lze kromě číselných souřadnic na ploše použít i takzvané kotvy (anchors), které definují spojení určité části prvku s částí jiného prvku. Lze takto například ukotvit levou a pravou stranu prvku na korespondující strany „rodičovského“ prvku. Výsledkem této operace je „potomek“, který má vždy stejnou šířku jako jeho „rodič“, a to i v případě změny velikosti rodiče během běhu aplikace. Dále je možné provázat *signály* (oznámení, že došlo k události) a *sloty* (reakce na události) mezi různými prvky a tím specifikovat základní chování rozhraní.

Vzhled rozhraní je zcela v režii tvůrce. Veškeré objekty na ploše lze ovlivňovat pomocí podmnožiny pravidel jazyka CSS. Tyto jsou pak dále manipulovatelné jak JavaScriptem, tak případně z rozšiřujících tříd C++. Díky tomu je možné snadno vytvářet například barevné přechody nebo animace.

Mezi hlavní výhody tohoto jazyka patří jeho podobnost s webovými technologiemi. Je tedy možné vytvářet dynamická uživatelská rozhraní velmi podobným způsobem, jakým se tvoří webové stránky, a není nutné učit se zcela nový systém. Díky multiplatformnosti frameworku Qt je také výsledná aplikace přenositelná na velké množství platforem – podporován jsou mimo jiné mobilní operační systém Android, MS Windows a X Window Server, který je typicky používán v různých linuxových distribucích.

Nevýhodou při využití QML je skutečnost, že jde o jazyk a nikoliv o programovou knihovnu. Do aplikace je proto nutné zabudovat kompletní překladač tohoto jazyka a velmi pravděpodobně také definice základních prvků, tedy jakousi „standardní knihovnu“. Také systém zpracování událostí od uživatele je úzce vázán na systém frameworku Qt, využívající jeho vlastní implementaci signálů a slotů. Výsledná aplikace je pak velmi ovlivňována tímto frameworkem, což nemusí být žádoucí.

Jak jazyk QML, tak framework Qt jsou detailně zdokumentovány a dokumentace je volně dostupná na webu[1], další zdroje jsou dostupné na oficiální stránce Qt projektu[2]. Vzhledem k rozšířenosti frameworku a existují komunitě okolo není velký problém sehnat v případě nutnosti pomoc na fórech nebo na jiných komunikačních kanálech. Zdrojové kódy pro vývoj jsou dostupné buď volně pod licencí GNU (L)GPL verze 2.1 nebo 3, nebo pod placenou komerční licencí. Je tedy je možné použít jak zdarma pro open-source aplikace, tak za poplatek pro uzavřené aplikace.

Jazyk QML je tedy použitelný zejména pro mobilní aplikace, které od začátku počítají s použitím tohoto jazyka a přidruženého frameworku. Na druhou stranu je nutné vývoj přizpůsobit specifikacím frameworku Qt, a může se stát, že spíše než k využívání prvků frameworku pro potřeby aplikace dojde k úpravám aplikace pro potřeby frameworku. Není to tedy vhodné řešení pro již existující projekty, které pouze potřebují přidat uživatelské rozhraní.

CEGUI

CEGUI je zkratkou pro *Crazy Eddie's GUI System*. Jde o svobodnou knihovnu poskytující prostředky pro tvorbu „oken" a grafických prvků pro grafické knihovny a rozhraní, které tyto prostředky nenabízejí nebo nabízejí pouze v omezené podobě. Je napsaná v jazyce C++ za použití objektově orientovaného návrhu. Dle vlastního popisu je „zaměřena na vývojáře, kteří by měli trávit svůj čas tvorbou skvělých her, nikoliv programováním GUI subsystémů" [3].

Uživatelské rozhraní vykreslované touto knihovnou lze vytvořit buď postupným vytvářením a provazováním C++ objektů v kódu aplikace, nebo vytvořením sady XML dokumentů, které požadované rozhraní popisují. V případě vytváření pomocí XML souboru je v aplikaci ještě nutné provést navázání jednotlivých událostí na příslušné reakce. V obou případech ale CEGUI využívá další XML soubory pro popis načítaných textur, definice stylů jednotlivých komponent, načítání fontů a obecně pro popis a načtení externích zdrojů.

Jak již bylo zmíněno, pro definici vzhledu se používají XML „stylopisy", které definují vzhled pro jednotlivé prvky rozhraní. Tento přístup umožňuje odlaďovat vzhled aplikace bez zásahu do zdrojových kódů, a tedy je možné jej svěřit i návrháři bez znalostí programování. Oproti dříve popsanému QML tento způsob definice stále vyžaduje nastudování specifického formátu a delší čas při tvorbě stylu.

Výhodou popisu celého rozhraní pomocí XML souborů je možnost jejich úpravy bez překompilovávání celého projektu. Je tedy možné vytvořit kostru aplikace a vhled a formu rozhraní pilovat bez dalšího zásahu do zdrojových kódů. Tyto soubory také mohou být generovány nějakým obecnějším nástrojem, a je tedy možné vytvořit například WYSIWYG editory, jejichž výstupem je kompletní popis veškerých zdrojů rozhraním použitých, které stačí do aplikace načíst pomocí malého množství instrukcí. Tyto definice jsou také dále šiřitelné bez nutnosti poskytovat zdrojový kód zbytku aplikace.

Nevýhodou této knihovny je ale závislost na jmenovaných XML souborech. Pokud již aplikace obsahuje systém načítání textur pro svoji vlastní činnost, bohužel není možné tyto textury snadno „recyklovat" pro použití v uživatelském rozhraní. Je sice možné danou texturu popsat XML definicí, ale výsledkem je pouze dvojité načtení daného zdroje – jednou pro potřeby rozhraní, podruhé pro potřeby aplikace samotné. Knihovna v sobě také poměrně logicky obsahuje syntaktický analyzátor jazyka XML, který je v případě statického linkování zahrnut i do výsledné aplikace. Je-li snahou programátora spustitelný soubor o co

[1] http://doc.qt.io
[2] http://qt-project.org

nejmenší velikosti (například pro mobilní zařízení s malou pamětí), může být tato „zbytečná“ komponenta překážkou.

CEGUI je dodávána spolu s dokumentací vytvořenou programem *Doxygen*. V dokumentaci je popsáno zejména rozhraní knihovny, ale obsahuje i výukové tutoriály, popisy doporučených postupů a specifikace formátů používaných XML dokumentů.

Tato knihovna je zaměřená spíše na hry pro tradiční stolní počítače. Mezi podporované operační systémy patří MS Windows, Apple OS X a různé distribuce GNU/Linuxu. Podpora mobilních operačních systémů, jako je například Android, bohužel není zahrnuta ani plánována.

Jako open-source knihovna je CEGUI volně dostupná ke stažení na webu [3]. Ve starších verzích (do verze 0.4.1) je dostupná pod GNU LGPL verze 3, od verze 0.5 dále je dostupná pod MIT licencí. Je ji tedy možné použít jak v open-source, tak v uzavřených aplikacích zcela zdarma.

CEGUI je stále poměrně velká knihovna, cílená na hry pro klasické stolní počítače. Pro takové hry, kterým nevadí oddělení zdrojů pro rozhraní a samotnou hru, může jít o zajímavé řešení implementace grafického uživatelského rozhraní. Pokud je ale hra vyvíjena s tím, že ji bude možné spustit i na dotykových a mobilních zařízeních, nebo již obsahuje vlastní systém načítání zdrojů a je žádoucí tento systém použít i pro uživatelské rozhraní, není tato knihovna vhodným řešením.

GWEN, turbobadger

GWEN (*GUI Without Extravagant Nonsense*) a turbobadger jsou malé knihovny pro tvorbu uživatelských rozhraní, navržené s cílem neplést se tvůrci do jeho programovacího stylu, a pouze mu poskytnout základní mechanismy a logiku pro vykreslování jednotlivých grafických prvků. Mají pouze minimální závislosti a snaží se dokonce být nezávislé na pokročilých vlastnostech kompilátoru, a tím i o maximální přenositelnost.

Tvorba rozhraní probíhá přímo ve zdrojovém kódu aplikace postupným vytvářením jednotlivých grafických komponent a jejich propojování do funkčního celku. U obou knihoven je použit model stromu, kdy je celé rozhraní obsaženo v jednom kořenovém „okně“ a jednotlivé prvky jsou pak vykreslovány pouze v rámci tohoto „okna“. Tento způsob mimo jiné umožňuje zavést samostatný souřadnicový systém, nezávislý na souřadnicích používaných při vykreslování zbytku scény. Také je díky tomu možno manipulovat rozhraním jako jiným běžným objektem ve scéně a vytvářet tak různé zajímavé efekty.

Vzhled rozhraní je u obou knihoven definován hlavně pomocí bitmapových obrázků. Zatímco ale knihovna GWEN používá jednu jedinou bitmapu, kterou si poté vnitřně „rozřeže“ a jednotlivé části používá pro vzhled různých prvků, knihovna turbobadger žádné „rozřezávání“ neprovádí a pro každý druh prvků rozhraní používá zvláštní bitmapu.

Bohužel obě knihovny sdílejí jednu velkou nevýhodu a překážku v jejich nasazení. Ani pro jednu z nich není dostupná prakticky žádná dokumentace. Jediný zdroj informací o těchto knihovnách je jejich zdrojový kód a demonstrační příklady, distribuované s tímto kódem. Je proto velmi těžké s nimi začít pracovat. Díky neexistenci dalších informačních zdrojů také není vůbec jasný stav a budoucnost těchto projektů. U knihovny turbobadger lze z historie revizí ještě usuzovat, že projekt je aktivně vyvíjen, ale v případě projektu GWEN je poslední revize již staršího data a není vůbec jasné, zda autor nebo autoři hodlají v jejím vývoji a údržbě pokračovat.

[3] http://cegui.org.uk

Obě knihovny, jak GWEN[4], tak turbobadger[5], jsou dostupné na internetu. Zdrojové kódy oubou knihoven jsou dostupné pod svobodnou licencí, GWEN pod licencí MIT, turbobadger pod licencí zlib. Je tedy legálně možné je zdarma použít v open-source i uzavřených aplikacích.

Jak již bylo zmíněno výše, tyto knihovny cílí na vývojáře, kteří nechtějí přizpůsobovat svůj styl programování externím knihovnám, ale chtějí pouze použitelnou implementaci GUI logiky. Bohužel je snaha autorů podkopána neexistencí rozumnější dokumentace, což činní tyto knihovny relativně těžce použitelnými. V případě jejího doplnění může jít o zajímavé alternativy k větším knihovnám, ale v současné době nejsou pro produkční aplikace vhodnými řešeními.

Guichan

Guichan je malá a efektivní GUI knihovna zaměřená pro tvorbu uživatelských rozhraní ve hrách. Podobně jako předešlé dvě knihovny je poměrně minimalistická a snaží se tvůrci aplikace co nejméně plést do stylu programování. Je ovšem starší než ony a na jejím návrhu a struktuře lze vypozorovat větší využívání pokročilejších vlastností kompilátoru a standardní knihovny. Z těchto vlastností lze usuzovat, že vlastnosti a omezení mobilních zařízení nebyly při implementaci knihovny zohledňovány.

Rozhraní je opět tvořeno pomocí instancování objektů představujících jednotlivé prvky rozhraní a jejich spojování do funkčního celku. Opět je zde uplatněn model jednoho kořenového „okna“, které funguje jako omezení plochy, na které se bude rozhraní vykreslovat, a také je hlavním spojovníkem mezi grafickým rozhraním a aplikací samotnou.

Implementace vzhledu je v základu velmi jednoduchá – lze určit barvu pozadí a popředí (s konkrétním efektem závislým na konkrétním prvku). Není implementován žádný způsob hromadnějšího nastavování vzhledu, vše je ponecháno na tvůrci aplikace. Také, ačkoliv je k dispozici načítání a zobrazování obrázků, není možné načtené obrázky použít pro definici vzhledu (například použít obrázek jako pozadí prvku).

Důvodem pro takto jednoduché možnosti definice vzhledu je opět snaha neplést se autorovi aplikace do stylu programování. Guichan totiž předpokládá a podporuje dědění z poskytnutých tříd a přetěžování jejich členských metod. Je tedy možné implementovat vlastní způsob úpravy vzhledu a vykreslování prvků, který bude bez problémů možné používat se zbytkem prvků knihovnou poskytnutých. Další výhodou, kterou tento přístup přináší, je transparentnost knihovny. Většina vnitřní logiky používá veřejné metody a není zde mnoho „magických“ míst, jejichž implementace by závisela na konkrétní implementaci ostatních částí knihovny.

Stejná minimalističnost může být ovšem i nevýhodou. Kromě velmi jednoduchých rozhraní není Guichan použitelný ihned, bez implementace vlastních zobrazovacích metod. Také může nepříjemně překvapit chybějící propojení některých základních událostí. Jako příklad lze uvést třídu `ScrollArea`, která implementuje zobrazování prvků na posuvné ploše. Při jejím použití je nutné „ručně“ navázat události posunu plochy a také poskytnout implementaci logiky pro zjištění relativní pozice posuvníku a jeho velikosti. Pravděpodobně jde opět o důsledek snahy přenechat autorovi aplikace co nejvolnější ruce, ale protože logika posuvné plochy bude v drtivé většině případů velmi podobná, není její neexistence zcela očekávána.

Dokumentaci pro knihovnu Guichan tvoří primárně popis jejího aplikačního rozhraní

[4] https://github.com/garrynewman/GWEN
[5] https://github.com/fruxo/turbobadger

(API), který je generován pomocí programu *Doxygen* a lze jej opět nalézt na internetu[6]. Oficiální web projektu je bohužel dlouhodobě mimo provoz, dostupná je pouze jeho starší verze[7]. Tato starší verze ale stále obsahuje relevantní informace a lze ji používat pro referenci. Pro kontakt s vývojáři je dále dostupných několik e-mailových konferencí, které lze také nalézt na dostupném webu.

Jak již bylo zmíněno v úvodu, Guichan pravděpodobně nebyl navrhován pro použití na mobilních zařízeních. V případě použití vhodných implementací standardní knihovny pro mobilní zařízení však pravděpodobně není důvod, proč by na nich tato knihovna nebyla použitelná.

Knihovna by měla být dostupná na serveru SourceForge, konkrétně na výše zmíněných adresách. Díky nefunkčnosti aktuálního webu [8] tam lze nalézt bohužel pouze starší verze. Knihovna je dostupná pod revidovanou BSD licencí (*3-clause BSD*), a je tedy volně použitelná jak v open-source, tak uzavřených aplikacích.

Na rozdíl od předchozích dvou popsaných „minimalistických knihoven“ je Guichan použitelná i pro vážnější vývoj, zejména díky existujícím návodům pro začátečníky a dokumentaci API. Dlouhodobě nefunkční projektové stránky ovšem vzbuzují otázku, zda je projekt stále ještě vyvíjen a udržován, a zda-li by nebylo lepší porozhlédnout se po jiné variantě. Také na mobilních zařízeních může být její nasazení poněkud problematické. Pro desktopové aplikace, zejména ty, které nevyžadují složitější rozhraní, však zůstává použitelnou variantou.

[6] http://guichan.sourceforge.net/api/
[7] http://guichan.sourceforge.net/oldsite/
[8] http://guichan.sourceforge.net

Kapitola 3

Návrh knihovny

Tato kapitola se zabývá návrhem nové věstavitelné GUI knihovny, použitelné jak na běžném PC, tak na mobilních zařízeních. V první části jsou rozebrány základní principy, kterými se návrh řídí, a jejich efekt na výslednou podobu návrhu.

Ve druhé části jsou rozebírány jednotlivé oblasti, pokryté touto knihovnou. V každé oblasti jsou rozebírány problémy spojené s danou oblastí a následně jejich navržená řešení.

Poslední část se zabývá výslednou navrženou strukturou tříd a jejich vztahy. Jsou zde popsány jednotlivé prvky rozhraní, jejich účel a zamýšlené využití.

Kapitola primárně čerpá z popisů knihoven SDL2 a OpenGL, jejich společných vlastností a rozdílů [7, 6].

3.1 Základní principy

Při návrhu struktury knihovny jsem si stanovil několik principů, které musí výsledná knihovna zohledňovat. Účelem stanovení těchto principů je zajištění konzistence jednotlivých dílčích částí knihovny, a zároveň specifikace rolí a chování těchto částí v rámci celé knihovny.

Princip modularity

Ačkoliv primárním cílem této knihovny je poskytnout nástroje pro tvorbu uživatelských rozhraní nad knihovnou SDL2, pro další vývoj a rozšíření této knihovny je důležité, aby na knihovně SDL2 nebyla zcela závislá. Kvůli tomuto požadavku je nutné poskytnout uživateli možnost nahradit knihovnu SDL2 jinou knihovnou nebo částí knihovny, která poskytuje podobné prostředky.

Modularita v tomto případě znamená právě možnost záměny implementace části knihovny za jinou implementaci, která ovšem poskytuje stejnou funkčnost. Tuto změnu musí být možné provést bez jakéhokoliv zásahu do existujícího kódu knihovny.

Příkladem takové záměny může být výše zmíněné nahrazení knihovny SDL2 například přímým používáním prostředků operačního systému pro tvorbu oken[1].

Důsledkem dodržování tohoto principu by měla být nezávislost navržené knihovny na jediné konkrétní implementaci používaných prostředků.

[1] X Window v GNU/Linuxu, WinAPI pod MS Windows a podobně.

Princip rozšiřitelnosti

Tento princip úzce souvisí s předchozím principem. Aby totiž bylo možné modularitu knihovny plně využít, musí být implementováno více zaměnitelných modulů, poskytujících daný prostředek či funkčnost. Protože je ale implementace více takových modulů mimo rozsah této práce, je velmi žádoucí usnadnit tvorbu těchto modulů v budoucnu. Také je vhodné umožnit snadnou implementaci zcela nových prvků knihovny, zejména ze strany případných uživatelů této knihovny.

Rozšiřitelnost tedy v tomto případě znamená možnost přidání zcela nových implementací ke stávajícím možnostem knihovny, opět bez nutnosti zasahovat do implementací existujících. Může jít jak o přidání například modulu poskytujícího nový způsob vykreslování prvků knihovny, tak o tvorbu zcela nových prvků rozhraní a podobných rozšíření.

Díky dodržování tohoto principu by výsledná knihovna měla být nezávislá na úsilí jediného autora, a mělo by být relativně snadno přizpůsobitelná požadavkům jednotlivých uživatelů.

3.2 Obecná struktura knihovny

Primární účelem knihovny, vytvořené v rámci této práce, je doplňovat existující aplikace a poskytovat jim implementaci různých prvků grafického uživatelského rozhraní. Nemá ale tvořit samostatnou část aplikace. V ideálním případě by měla fungovat jako přirozená součást aplikace schopná komunikace s ostatními jejími částmi.

Pro dosažení tohoto efektu je navržená knihovna koncipována jako sada různých prvků, modulů a podobných objektů. Tyto lze v aplikaci použít jak jednotlivě, tak v podobě zkombinovaných bloků, skládajících se z několika spolupracujících objektů.

Poskytované objekty lze z pohledu uživatele-programátora rozdělit do tří hlavních kategorií. První kategorii tvoří samotné ovládací prvky. Do druhé kategorie spadají objekty umožňující vlastní vykreslení prvků uživatelského rozhraní. Do třetí kategorie patří objekty zajišťující komunikaci prvků uživatelského rozhraní z první kategorie s vrstvou aplikace spravující okno či vykreslovací plochu aplikace.

Každá kategorie má svá specifika, která je třeba při návrhu zohlednit, a jsou proto rozebrány zvlášť v následujících sekcích. Objekty v jednotlivých kategoriích ale nejsou zcela odlišné. Jejich hlavní společnou vlastností je schopnost spolupracovat s jinými objekty, jak ze své, tak z jiné kategorie. Díky této vlastnosti dokáže knihovna fungovat jako celek a zároveň zůstává dostatečně modulární.

Prvky grafického uživatelského rozhraní

První výše zmíněnou kategorii tvoří objekty modelující jednotlivé prvky uživatelského rozhraní. Účel těchto objektů je dvojí: uchovávání aktuálního stavu rozhraní a poskytnutí implementace základní logiky chování běžných prvků uživatelského rozhraní, nezávislou na konkrétním mechanismu vykreslování či zpracování událostí.

Při návrhu způsobu uchování stavu rozhraní je třeba řešit zejména možnosti přístupu k uchovávaným informacím. V průměrné aplikaci se počty použitých prvků rozhraní mohou pohybovat až v řádu desítek vzájemně se ovlivňujících objektů. Ke každému z nich musí být možné kdykoliv přistoupit a získat informace o jeho stavu. Také musí být možné dynamicky měnit skladbu celého rozhraní přidáváním nových nebo odebíráním existujících prvků.

Dalším problémem, který je třeba zohlednit, je tvorba nových prvků rozhraní. Nově naprogramované prvky musí být možné bezbolestně integrovat mezi prvky stávající, aby nebylo nutné kvůli každému novému prvku upravovat vykreslování a podobné ne zcela související části knihovny. Také je vhodné umožnit skládání jednodušších prvků do komplexnějších celků tak, aby bylo s výsledkem možné pracovat jako s jediným prvkem.

Dalším faktorem, jehož řešení je nutné nalézt, je vyžadovaná modularita knihovny. Libovolný prvek musí být vykreslitelný libovolnou implementací vykreslování a musí být chopen reagovat na požadavky aplikace a jejího uživatele nezávisle na konkrétní implementaci komunikační vrstvy.

Je také potřeba myslet na předpokládaný způsob vykreslování prvků do 2D prostoru. Používané prvky se totiž mohou překrývat, a tak je důležité uchovávat i jejich relativní pozici na ose kolmé k ploše aplikace. Tato informace slouží pro rozhodnutí, který prvek bude na daném místě nakonec vykreslen, pokud dojde k překryvu.

Navrženým řešením problémů přístupu k prvkům, jejich dynamického přidávání a odebírání, skládání prvků a uchování informace o relativní „hloubce“ prvků je jejich uchování v datové struktuře typu strom. Tento způsob při dodržení určitých konvencí poměrně elegantně řeší všechny vyjmenované problémy.

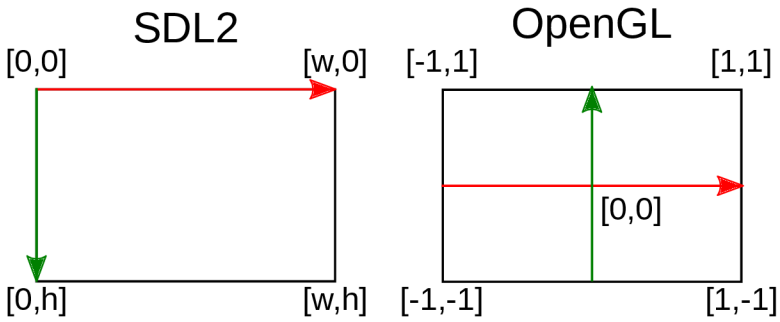
Pro vyhledání prvku je nutné, aby byl hledaný prvek jednoznačně identifikovatelný, ideálně pomocí nějakého číselného nebo textového označení. Pokud by možnost vyhledání prvku byla jediným požadavkem na strukturu prvků, nabízela by se možnost použít vyvážený binární vyhledávací strom, s použitím označení prvku jako řadícího klíče. Protože ale jediným požadavkem není, a obecnější stromová struktura má i jiné vlastnosti využitelné pro řešení dalších problémů, je použití obecnějšího stromu vhodnější. V obecném stromu lze jednoduše implementovat vyhledávání pomocí definice vhodného průchodu přes všechny prvky daného stromu. V tomto případě jde například o hloubkový průchod (*depth-first*). Na strom lze pak nahlížet jako na sekvenci prvků a vyhledávat sekvenčně. Ačkoliv nejde o ideální vyhledávací algoritmus, pro potřeby této knihovny je dostačující[2], a tento způsob průchodu stromem je použitelný i u dalších problémů.

Pro skládání a dynamickou správu prvků je strom téměř ideálním řešením. Skládáním prvků totiž vzniká stromová struktura poměrně přirozeně (složený prvek je kombinací svých částí, z nichž každá může být také složeným prvkem). Výsledkem aplikace tohoto pohledu na všechny uzly stromu vzniká možnost se složeným prvkem pracovat buď jako s celkem, nebo jako se součástí a podstromem většího stromu prvků. Při dynamické práci s prvky lze pak přidávat či odebírat celé podstromy nebo jen jednotlivé prvky, přičemž je vždy zachována struktura zbývajících částí stromu.

Z pohledu vykreslování je důležitá definice sekvenčního průchodu stromem. Při použití hloubkového průchodu je každý prvek definován i svým pořadím v iteraci a vykreslování je pak konzistentní. Složené prvky nevyžadují zvláštní zacházení, neboť při tomto způsobu průchodu stromem je vždy zpracován celý podstrom před tím, než je zpracován libovolný prvek dalšího podstromu na stejné úrovni. Díky tomu nedochází k „rozsypání“ složených prvků, které jsou vždy zpracovány jako jeden celek.

Co se týče problémů tvorby zcela nových prvků a modularity celého systému, zde se jako vhodné řešení nabízí využít dědičnosti objektů. Při definování vhodného předka společného všem prvkům rozhraní pak lze k těmto prvků přistupovat uniformně, bez nutnosti definice nějakých zvláštních případů. Specifika různých prvků pak lze zohlednit definicí vhodného rozhraní a použití virtuálních metod. Díky tomu může libovolný potomek poskytnout vlastní

[2]Pokud by se v praxi ukázal jako nedostatečný, lze implementovat jiný způsob vyhledávání.



Obrázek 3.1: Srovnání souřadnicových systémů v knihovně SDL2 a ve specifikaci OpenGL

implementaci potřebných částí (například instrukce pro vykreslení) bez nutnosti úprav objektů pracujících se společným předkem.

Navrženým řešením této části knihovny je tedy implementace společného předka všech poskytovaných prvků rozhraní. Účelem tohoto předka je zejména definice rozhraní, používaného ostatními částmi této knihovny, a poskytnutí funkčnosti umožňující uchovávat prvky rozhraní ve stromové struktuře. Nad touto strukturou je nutné definovat hloubkový průchod a vyhledání libovolného jejího prvku pomocí jeho označení.

Vykreslování rozhraní

Tvorbu grafické reprezentace rozhraní lze považovat za primární účel celé knihovny. Ačkoliv logika chování či interaktivita prvků jsou také důležité, bez grafického výstupu pozbývají svého významu. Grafická reprezentace je také nejpravděpodobnějším cílem uživatelských úprav a modifikací, neboť vzhled rozhraní je nutné sladit se zbytkem aplikace, která může mít mnoho různých podob. Je tedy nebytné, aby bylo možné tyto úpravy provádět s co nejmenším úsilím.

Na druhou stranu je ale nutné stanovit některé základní konvence, aby bylo možné poskytnout společný základ pro vykreslování. Jednou z těchto konvencí je souřadnicový systém. Různé vykreslovací knihovny a nástroje používají různé variace souřadnicových systémů, a bohužel neexistuje jednotný standard. Jako příklady různých možných systémů souřadnic lze uvést knihovnu SDL2 a specifikaci OpenGL.

Knihovna SDL2 používá dvourozměrný kartézský souřadnicový systém, kterým přímo popisuje pozici v rámci plochy aplikace. Počátek souřadnicového systému je umístěn v levém horním rohu plochy aplikace a hodnoty na vertikální ose rostou směrem k jejímu dolnímu okraji. Souřadnice jsou celočíselné a představují vzdálenost od počátku v obrazových pixelech.

OpenGL naproti tomu používá trojrozměrný souřadnicový systém, popisující pozici v obecném trojrozměrném prostoru. Pozice objektu na ploše aplikace je následně z této pozice vypočítána za běhu aplikace a závisí na více faktorech. Pro účely srovnání se dále budu zabývat popisem souřadnicového systému, do kterého je pozice v prostoru promítnuta, jsou-li veškeré parametry tohoto promítání ponechány ve výchozím stavu. V takovém případě je počátek souřadnicového systému umístěn ve středu plochy aplikace a hodnoty na vertikální ose rostou směrem k jejímu hornímu okraji. Jednotkový vektor nemusí mít ve všech směrech stejnou velikost, protože závisí na rozměrech plochy aplikace. V případě rovnoběžnosti s horizontální nebo vertikální osou souřadnicového systému je jeho velikost rovna vzdálenosti od počátku systému k příslušnému okraji plochy[6, kap. 4].

Pro větší názornost lze grafické srovnání obou systémů nalézt na obrázku 3.1. V obou případech jde o schéma plochy aplikace, v případě OpenGL tedy již po převodu souřadnic ze 3D prostoru. Šipky představují jednotlivé osy a směr růstu jejich hodnot. Uvedené souřadnice popisují souřadnice příslušných rohů plochy aplikace[3].

Jak je vidět, uvedené souřadnicové systémy jsou velmi odlišné, a existuje i množství dalších možných způsobů popisu pozice v okně aplikace. Je tedy nutné stanovit jednotný systém pro vnitřní použití v navržené knihovně, a ten v případě nutnosti při vykreslování převést do souřadnicového systému použitém v daném vykreslovacím systému.

Dalším problémem je interakce s vykreslováním zbytku aplikace. Proces vykreslování prvků rozhraní navržené knihovny nesmí za žádných okolností ovlivňovat vykreslování dalších elementů aplikace, aby nedocházelo ke změnám uživatelem používaného prostředí. Je také nutné zajistit vykreslení celého rozhraní najednou, aby nedocházelo k „promíchání“ grafických výstupů uživatelského rozhraní a jiných prvků aplikace.

Třetím problémem relevantním pro vykreslování je získávání a ukládání obecných obrazových dat. Opět zde hrají roli prostředky poskytované jednotlivými mechanismy vykreslování. Každý mechanismus má své datové struktury, které popisují uložená obrazová data, a většinou vyžaduje, aby jím zpracovávaná data byla v těchto strukturách uložena. Je tedy opět nutné poskytnout nějaké společné rozhraní které umožní převádět uložená data mezi jednotlivými strukturami a formáty.

Při návrhu prvotního získávání těchto dat je opět třeba mít na paměti předpokládané použití této knihovny jako komponenty ve větší aplikaci. Lze předpokládat, že tato aplikace již implementuje nějaký způsob načítání obrazových dat z disku do paměti aplikace. V takovém případě je krajně nevhodné nutit uživatele této knihovny k opětovnému načítání těchto dat z disku. Ačkoliv tedy navržená knihovna může poskytnout nástroje pro načtení externích souborů, musí být schopna použít i data, která se již v paměti aplikace nacházejí.

Posledním zkoumaným problémem je způsob vykreslení textového řetězce. Tento problém je poměrně komplexní, protože v sobě zahrnuje více různých podproblémů. Mezi ně patří například načítání fontu (ať již bitmapového, nebo vektorového), výpočet velikosti písma, různé varianty jednotlivých symbolů a další. Vlastní řešení všech těchto problémů je zcela mimo zaměření a rozsah této práce. Další možností je implementovat základní podporu pro jednoduché bitmapové fonty, nebo využít externí knihovny pro vykreslování textu pomocí vektorových fontů a poskytnout funkce umožňující s touto knihovnou pracovat.

Oba přístupy mají své výhody a nevýhody. Pro bitmapové fonty hovoří hlavně možnost použití prakticky libovolných obrazových dat jako fontu a jednodušší implementace bez dalších externích závislostí. Proti hovoří zejména nutnost vytvoření fontu uživatelem (nebo uzpůsobení existujícího tak, aby vyhovoval navržené knihovně), a obecně horší vzhled výsledného textu. V případě podpory vektorových fontů je výhodou dobrá dostupnost existujících fontů, lepší vzhled vykresleného textu a obecně méně nutné práce ze strany uživatele navržené knihovny. Nevýhodami mohou být nutnost přidání externí závislosti navržené knihovny na další knihovně a v případě použití vlastního fontu větší obtížnost jeho tvorby.

Navrženým řešením problému s obecným vykreslováním je poskytnutí sady rozhraní, popisujících možnosti práce s obecným vykreslovacím mechanismem, a alespoň jedné referenční implementace tohoto rozhraní. Díky tomu zůstane zachována modularita knihovny a zároveň lze použít optimalizace specifické pro daný vykreslovací mechanismus.

Součástí rozhraní musí být i specifikace používaného souřadnicového systému. Protože se navržená knihovna zabývá vykreslováním na dvourozměrnou plochu, byl zvolen souřad-

[3]Konstanty `w` a `h` představují šířku, respektive výšku plochy aplikace v pixelech.

nicový systém používaný knihovnou SDL2. Důvodem je hlavně jeho poměrná intuitivnost, spočívající ve specifikaci pozice a velikosti prvku přímo za použití pixelů na ploše aplikace.

Problém případného promíchání prvků rozhraní se zbytkem aplikace je v tomto návrhu řešen definicí obecné metody rozhraní, která vykreslí celý strom prvků během jediného volaní. Uživatel knihovny tak má možnost snadno vykreslit celé rozhraní na takovém místě v kódu aplikace, kdy nebude k promíchání docházet.

Protože jednotlivé vykreslovací mechanismy mají své preferované metody skladování obrazových dat, je vhodné ponechat práci s těmito daty přímo na nich. Navrhovaným řešením tak je opět poskytnutí obecného rozhraní pro práci s obrazovými daty a jejich formátem, a implementace tohoto rozhraní optimalizovaná pro použití s příslušným vykreslovacím mechanismem.

Co se týče vykreslování textu, byla zvolena varianta podpory vektorových fontů s pomocí externí knihovny. Důvodem je právě větší uživatelská přívětivost. Pro zjednodušení implementace a méně vyžadovaného nastavování před použitím navržené knihovny není spolupráce s touto externí knihovnou implementována ve formě modulu, ale jako pevná závislost. Protože tato skutečnost ale porušuje princip modularity, je nutné zvolit co nejpřenositelnější knihovnu. V případě potřeby pak i tuto část navržené knihovny převést na vlastní skupinu zaměnitelných modulů.

Celkové navržené řešení této části knihovny se tedy sestává z definice programového rozhraní pro vykreslování prvků a ukládání obrazových dat, a implementace těchto rozhraní nad zvolenou metodou vykreslování. Druhou částí je poskytnutí objektů pro převod textových řetězců na obrazová data s použitím vektorových fontů. Implementace těchto objektů používá externí knihovnu a musí mít takovou podobu, aby bylo možné ji modularizovat.

Komunikace se zbytkem aplikace

Poslední částí návrhu knihovny je komunikace se zbytkem aplikace. Tato komunikace zahrnuje především mechanismus zachytávání relevantních událostí a reakcí na ně. Také sem spadá návrh způsobu specifikování uživatelských reakcí na jednotlivé události.

Jak již bylo zmíněno v úvodním popisu principů, jedním z cílů tohoto návrhu je zachovat knihovnu modulární a umožnit volbu z několika alternativ v každé oblasti. I když tedy zadání specifikuje pro komunikaci se zbytkem aplikace použít knihovnu SDL, je třeba navrhnout obecný mechanismus komunikace a zachytávání událostí, aby bylo možné eventuálně implementovat práci s dalšími podobnými knihovnami.

Při reagování na události se objevuje podobný problém jako při vykreslování – každý mechanismus pro tvorbu oken a správu událostí (dále jen „správce oken“) používá své metody a datové struktury pro zasílání informací do aplikace. Má-li si navržená knihovna zachovat modularitu i v této oblasti, je opět nutné vytvořit obecné rozhraní a převádět události od správce oken na tento vnitřní formát.

Zasílané události lze dále rozdělit na tři větší kategorie. Za prvé jde o události, které se vztahují ke konkrétní pozici na ploše aplikace a mohou být namířeny na jeden konkrétní prvek rozhraní (například kliknutí myši). Do druhé kategorie spadají události, které ovlivňují celou plochu aplikace a rozhraní na ně musí reagovat jako celek (například změna velikosti okna a plochy aplikace). Třetí kategorii tvoří ty události, které se rozhraní vůbec netýkají a je třeba přenechat je ke zpracování zbytku aplikace.

Události z různých kategorií je třeba zpracovávat různými způsoby. Události mířené na konkrétní prvek je třeba přenechat tomuto prvku a zpracování nechat na něm. Události ovlivňující celé rozhraní je třeba zpracovat přímo a upravit všechny ovlivněné objekty.

U událostí, které se rozhraní netýkají, je třeba indikovat, že nebyly zpracovány.

Pro události, mířené na konkrétní prvky, je třeba definovat příslušné reakce. Některé mohou být přímo vestavěné, ale aby měla tvorba rozhraní smysl, musí být uživatel schopen specifikovat vlastní reakce na různé události a změny daného prvku. Nejedná se ovšem o přímý překlad události zaslané správcem oken. Naopak jde o různé typy událostí specifické pro různé typy prvků rozhraní. Každý interaktivní prvek tak musí definovat, jaké akce a události podporuje, a poskytnout implementaci převádějící sled událostí správce oken na jednu nebo více událostí daného prvku.

Navržené řešení pro zachytávání událostí správce oken je podobné jako u vykreslování. Spočívá v definici programového rozhraní pro komunikaci s obecným správcem oken a zpracování jeho událostí, a implementaci tohoto rozhraní pro konkrétní správce oken. Součástí definice rozhraní je i definice datových struktur, představujících obecné události, které dokáže zbytek knihovny zpracovat a na které je potřeba převádět události konkrétního správce oken.

Kromě rozhraní pro zpracování událostí zapouzdřuje implementace výše zmíněného „komunikačního"rozhraní také práci s grafickým uživatelským rozhraním jako celkem. Měla by tedy obsahovat reference na všechny použité stromy prvků. Díky tomu je možné jednoduše reagovat na události ovlivňující celé rozhraní a upravovat potenciálně všechny existující prvky rozhraní.

Jako vhodný mechanismus pro definici uživatelských reakcí na události nad jednotlivými prvky jsem zvolil metodu zpětného volání funkcí („callbacks“). Tato metoda ponechává uživateli knihovny naprostou volnost při definici reakcí, a zároveň je rozumně implementovatelná bez použití specializovaných knihoven.

Navržené řešení problémů z této oblasti tedy spočívá v definici datových struktur pro obecné události, definici rozhraní pro práci s obecným správcem oken, a poskytnutí alespoň jedné implementace tohoto komunikačního rozhraní nad konkrétním správcem oken. Pro řešení definice uživatelských reakcí na události je použita metoda zpětného volání funkcí.

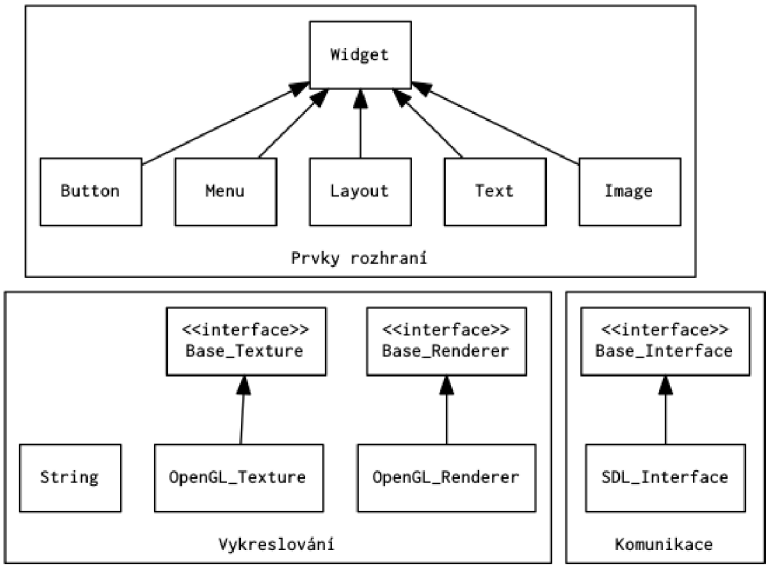
3.3 Základní sada funkčních prvků

Pro prvotní implementaci této knihovny byla navržena základní sada prvků rozhraní a zvoleny knihovny pro implementaci prvotních objektů pro vykreslování a komunikaci se zbytkem aplikace. Cílem této sady je poskytnout snadno rozšiřitelný základ, použitelný pro kompletní řešení jednoduchých rozhraní. Prvky v této sadě byly voleny tak, aby je bylo možné použít pro tvorbu rozhraní ovládaného myší nebo dotekem na dotekové obrazovce.

Volba použitých externích knihoven byla ovlivněna primárně zadáním této práce a pak také jejich dostupností na mobilních platformách. Pro komunikaci s aplikací je použita knihovna SDL2, kterou v oblasti vykreslování doplňuje specifikace OpenGL. Pro vykreslování textu pak byla vybrána knihovna FreeType.

Přehled navržené struktury knihovny lze nalézt na obrázku 3.2. Na tomto obrázku jsou zobrazeny navržené hierarchie tříd, nutné pro fungování základní sady. Nejde o přesný popis implementace, ale spíše o konceptuální skladbu knihovny.

Základním prvkem rozhraní je společný předek všech ostatních prvků rozhraní, na výše zmíněném obrázku označený jako `Widget`. Účelem tohoto prvku je poskytnout základní společnou funkčnost. Tato funkčnost zahrnuje zejména výše rozebírané skládání jednotlivých prvků do stromu a jejich pozdější vyhledávání, a dále poskytnutí možnosti obousměrné iterace nad tímto stromem. Také slouží jako definice společného programového rozhraní,



Obrázek 3.2: Návrh hierarchie tříd

které umožňuje průhledně pracovat s jakýmkoliv jeho potomkem, aniž by bylo nutné znát jeho konkrétní typ.

Účelem prvků `Text` a `Image` je vykreslování obsahu, který přímo nereaguje na události od uživatele. Jak již jejich název napovídá, `Text` slouží pro vykreslení textového řetězce a `Image` pro vykreslení obrazových dat. Jak je vidět na obrázku 3.3, jsou oba koncipovány jako dekorátory nízkoúrovňových vykreslovaných objektů (`String` a `Texture`, popsány dále) a také jako stavební bloky komplexnějších prvků.

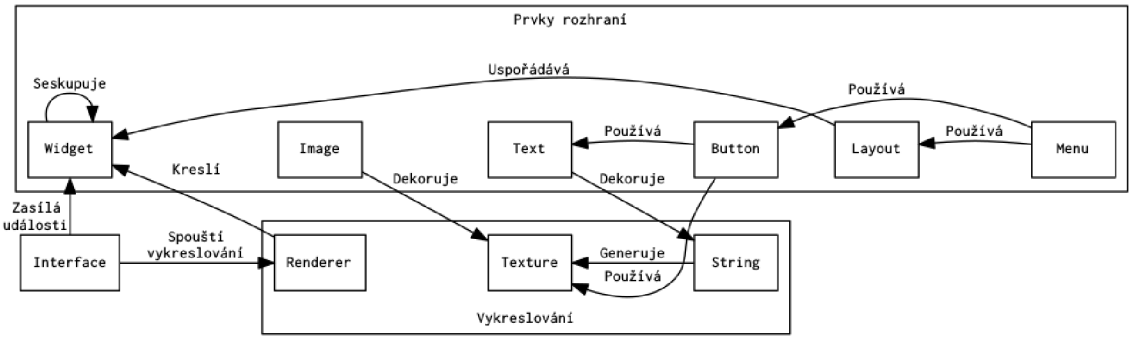
Prvek `Button` lze označit za opak předcházejících dvou prvků. Jde o obecný prvek, reagující na stisknutí tlačítka myši nebo dotyk na obrazovce. Jeho účelem je zpracovat posloupnost událostí zaslaných komunikační vrstvou, a provést uživatelem definované reakce. Mezi tyto reakce patří stisknutí tlačítka, jeho „zamáčknutí“ a jeho uvolnění. Je tedy možné použít jej jak jako klasické tlačítko (indikující stisk), tak jako přepínač mezi dvěma stavy.

Prvek `Layout`, neboli česky „uspořádání“, je prvkem usnadňujícím organizaci dalších prvků na ploše aplikace. Jeho účelem je automaticky uspořádávat tyto prvky do sloupce nebo řady, bez nutnosti specifikovat jejich přesnou polohu nebo rozměry. Jeho použití zahrnuje tvorbu různých sad souvisejících prvků, či automatické upravování rozměrů prvku v závislosti na jeho sousedech.

Prvek `Menu` slouží pro tvorby těch částí rozhraní, které nemají být vždy viditelné. Jeho úlohou je tvorba různých rozevíracích rolet s nabídkami či skrývání částí rozhraní za účelem uvolnění místa na ploše aplikace. Jde také o představitele skládaných prvků a slouží také jako reference pro jejich tvorbu. Jak je vidět na obrázku 3.3, pro svoji funkci využívá prvků `Button` pro přepínání viditelnosti a `Layout` pro automatické uspořádání skrývaného obsahu.

K vykreslení všech výše uvedených prvků rozhraní slouží objekty implementující rozhraní `Base_Renderer`, v tomto případě `OpenGL_Renderer`. Jeho účelem je zapouzdření volání nízkoúrovňových vykreslovacích funkcí do obecně použitelného rozhraní. Toto rozhraní pak umožňuje jediným voláním vykreslit celý strom prvků.

Zvláštní oblast vykreslování pokrývá rozhraní `Base_Texture` se svojí referenční implementací `OpenGL_Texture`. Jejich účelem je uchovávání obrazových dat v podobě optimalizo-



Obrázek 3.3: Vztahy mezi objekty základní sady

vané pro danou implementaci rozhraní `Base_Renderer`, přístupných přes obecné rozhraní. Spolupráce mezi těmito dvěma objekty zajišťuje co nejrychlejší vykreslení mnohdy objemných obrazových dat.

Posledním vykreslovacím objektem je `String`. Jeho účelem je převod textového řetězce z paměti aplikace na jeho grafickou reprezentaci s použitím vektorového fontu. Pro tento převod je použita již zmíněná knihovna FreeType, která generuje dvojrozměrná obrazová data pro jednotlivé znaky. Úkolem objektů třídy `String` tak je poskládat z těchto jednotlivých znaků objekt obsahující grafickou reprezentaci celého řetězce.

Programové rozhraní `Base_Interface` s referenční implementací `SDL_Interface` slouží pro zapouzdření komunikace s okenním manažerem (v tomto případě tedy knihovnou SDL2) do obecného rozhraní. Primárním účelem těchto objektů je překlad událostí z formátu okenního manažera do vnitřního formátu knihovny, který je zpracovatelný jednotlivými prvky rozhraní. Sekundárním účelem je pak usnadnění práce s rozhraním jako celkem. Napojením rozhraní na nějaký `Base_Renderer` je možné jednoduše zpracovávat události ovlivňující rozhraní jako celek (např.,změna velikosti okna aplikace). Připojením jednotlivých stromů prvků pak lze pohodlně zasílat události k příslušným prvkům a automaticky tyto stromy vykreslovat, což uživateli této knihovny tyto úkony výrazně zjednodušuje.

Kapitola 4

Implementace

Implementační kapitola se v první části věnuje specifikaci jazyka C++, použitého pro tvorbu knihovny a popisu způsobu řešení problémů způsobených jeho omezením na mobilních platformách. V další části jsou rozebrány jednotlivé prvky uživatelského rozhraní společně s popisem algoritmů použitých pro jejich implementaci.

Ve třetí části jsou popsány třídy spojené s vykreslováním rozhraní. Jsou zde popsány rozhraní jednotlivých tříd společně s jejich zamýšleným použitím. Dále jsou shrnuty technologie použité pro jejich implementaci. V poslední části je popsána komunikační vrstva knihovny se zbytkem aplikace.

Kapitola čerpá ze samotné implementace popisovaných částí, dále pak z popisu návrhových vzorů v [4] a metod generického programování v [1].

4.1 Popis výsledné knihovny

Knihovna vytvořená v rámci této aplikace patří mezi věstavitelné knihovny pro tvorbu rozhraní. Jde o objektovou knihovnu jazyka C++. Pro implementaci byla použita verze tohoto jazyka popsaná standardem C++11, což umožňuje použít novější vlastnosti tohoto jazyka, a také moderní verzi jeho standardní knihovny. Z této standardní knihovny jsou primárně používané definici abstraktních datových typů.

Zvláštní pozornost si zaslouží implementace statické identifikace konkrétních datových typů. Knihovna ve velké míře využívá principu dědičnosti a společných rozhraní pro práci s příbuznými datovými typy. V některých situacích je ovšem třeba identifikovat, jakého konkrétního typu je objekt, se kterým je právě operováno. Jazyk C++ pro tyto případy nabízí dynamickou identifikaci typů za běhu aplikace (*Run-time type identification, RTTI*). Na některých mobilních platformách však tento způsob není podporován, a bylo proto třeba implementovat jiné řešení.

Statická identifikace typů využívá specifikaci šablonových tříd, které poskytují informace o datovém typu během kompilace (*type_ traits*, [1]). Pro každou třídu, jejíž objekty je třeba identifikovat za běhu, je definována šablona, která se v závislosti na typovém parametru během kompilace rozvine na konstantní hodnotu unikátní pro danou třídu. Dále má každá taková třída definována virtuální metodu, která za běhu vrací právě tuto konstantní hodnotu. Protože v případě virtuální metody je vždy použita její varianta z odvozené třídy (i v případě, že je objekt považován za instanci bázové třídy), je vždy vrácen identifikátor odvozené třídy. Díky tomu se lze dotazovat na konkrétní typ objektu i bez použití RTTI.

4.2 Prvky rozhraní

Základem implementace všech prvků rozhraní je třída `widget`. Tato třída definuje společné rozhraní pro všechny ostatní prvky, a umožňuje manipulaci se společnými vlastnostmi. Každému prvku je tak možné nastavit barvu, pozici, velikost, viditelnost a textové označení pro snadnější vyhledání daného prvku ve větším stromu prvků.

Konkrétní interpretace těchto vlastností závisí na daném prvku – například barva u prvku `text` ovlivňuje barvu textu, zatímco u obecného prvku `widget` barvu, kterou má být vyplněna jeho plocha. Pozice prvku je definována jako pozice jeho levého horního rohu, a je vždy interpretována jako relativní k přímému předkovi ve stromě prvků, nebo k ploše aplikace (jde-li o kořen stromu). Po přidání sady prvků do stromu prvků tak zůstává zachována jejich vzájemná pozice.

Kromě společného rozhraní poskytuje třída `widget` i implementaci zmiňovaného stromu prvků a způsob iterace nad tímto stromem. Každý prvek si uchovává ukazatele na své přímé potomky v dynamickém seznamu, a ukazatel na svého přímého předka. Při připojení nového potomka tak přes něj získá daný předek přístup i k jeho potomkům, a lze takto vytvářet libovolně hluboké stromy. Ukazatel na předka pak slouží k procházení stromu směrem vzhůru, a postupnou iterací se lze dobrat až ke kořeni stromu.

Iterace nad stromem prvků je řešena implementací vlastních obousměrných iterátorů[4] ve třídě `widget::iter`. Tyto iterátory jsou vždy vázány na konkrétní strom prvků a uchovávají v sobě kompletní cestu od kořene stromu až k aktuálnímu prvku. Kromě iterace tak lze objektů této třídy využít jako unikátní identifikátor prvku v rámci daného stromu.

Implementace těchto iterátorů spočívá hlavně v uchování zásobníku iterátorů v seznamech potomků na jednotlivých úrovních stromu, a vhodné definici posunu vpřed a vzad. Iterační krok na iterátoru ve stromu se pak převádí na krok na vhodném iterátoru v seznamu potomků. Algoritmus tohoto převodu je shrnut v ukázce 4.1.

```
Node current_node;
Stack<Node> parents;

if current_node.has_children():
  parents.push(current_node);
  current_node := current_node.children.first();
else:
  current_node.next();
  while (current_node == parents.top.end()):
    current_node := parents.pop();
    current_node.next();
  endwhile
endif
```

Ukázka 4.1: Iterační krok ve stromu prvků

Tento algoritmus popisuje způsob hloubkového průchodu stromem za použití postupných iterací nad jednotlivými generacemi ve stromu. Jeho cílem je vždy navštívit aktuální prvek, poté „rekurzivně“ všechny jeho potomky a teprve poté se přesunout k jeho sourozenci. Podobně je definován i zpětný průchod stromem – nejdříve navštíví posledního potomka, poté všechny předešlé a teprve poté se přesune k rodiči.

První způsob iterace je využíván zejména při vykreslování, neboť tímto způsobem vykreslený potomek vždy překryje své předky. Zpětný působ iterace je využíván zejména při

propagování událostí. Protože uživatel vidí „nahoře“ posledního potomka, je nejdříve požádán o zpracování on, a pokud ke zpracování události nedojde, jsou postupně žádány předešlé prvky. Tímto způsobem je událost zaslána vždy k prvku, který ji dokáže zpracovat, a který je zároveň z pohledu uživatele „nejblíže“.

Statické prvky `image` a `text` jsou implementovány jako dekorátory tříd pro uchovávání textur a textových řetězců. K funkčnosti těchto tříd (popsané dále) přidávají vlastnosti společné pro všechny prvky rozhraní a umožňují jejich přidávání do stromu prvků. Prvek `text` navíc přidává možnost definici zarovnání textu v rámci větší plochy, jak horizontálně, tak vertikálně.

Prvek `button` je hlavním funkčním prvkem základní sady. Jeho implementace umožňuje zvolit si ze dvou různých chování při zachycení kliknutí myši nebo doteku. Lze jej použít jak jako klasické tlačítko (stisknuté jen po dobu držení tlačítka myši), nebo jako přepínač (kliknutí přepíná mezi stavy „sepnuto“ a „uvolněno“).

S těmito stavy a způsoby chování souvisí i specifikace reakcí na aktivitu tlačítka. Reakce `on_click` je spuštěna vždy při uvolnění tlačítka myši, nezávisle na způsobu použití či aktuálním stavu tlačítka – jde tedy o indikátor prostého klepnutí myši. Reakce `on_set` a `on_unset` se drobně liší v závislosti na způsobu použití. V případě klasického tlačítka indikují stisk, respektive uvolnění tlačítka myši (a v takovém případě jsou `on_click` a `on_unset` prakticky stejné). V případě přepínače indikují přepnutí do stavu „sepnuto“, respektive „uvolněno“.

Ke každé reakci přísluší seznam funkcí, které má daná reakce spouštět. V případě spuštění dané reakce jsou tyto funkce sekvenčně volány, a každé je předáno označení tlačítka, které danou funkci zavolalo. Je tak možné specifikovat libovolné množství reakcí na libovolnou kombinaci událostí, což umožňuje velmi flexibilní použití tohoto prvku při tvorbě rozhraní.

Schopností tříd odvozených od třídy `layout` je uspořádávání dalších prvků buď do slupce, nebo do řady, se zachováním konstantních rozměrů v opačném směru[1]. Libovolné změny velikosti prvku `layout` jsou pak aplikovány i na jeho přímé potomky, čímž jsou tito udržovány v konzistentním stavu.

Odvozené třídy tohoto prvku definují odlišné chování při přidávání nových potomků. Třída `expand_layout` při přidání nového potomka zvětší svoji plochu v daném směru tak, aby se nový potomek přesně vešel a pak jej umístí za předchozího potomka. Výsledkem tohoto chování je dynamicky rostoucí seznam prvků, kterým není třeba specifikovat přesnou polohu. Třída `fill_layout` naopak upravuje své potomky tak, aby vždy vyplnili celou její plochu. Po přidání nového potomka je plocha přerozdělena všem potomkům stejným dílem. Výsledkem tohoto chování je prvek, jehož plocha je vždy zcela vyplněna jeho potomky.

Poslední implementovaný prvek rozhraní představuje třída `menu`. Jejím účelem je zobrazovat a skrývat sadu dalších prvků na požádání uživatele. Její implementace spočívá ve využití prvku `layout` pro organizování skrývané sady a prvku `button` pro přepnutí mezi zobrazením a skrytím této sady. Skrývání a odkrývání je pak řešeno pomocí definice vlastní reakce na stisknutí přidruženého tlačítka, která přepne viditelnost skrývané sady.

4.3 Vykreslování

Základem vykreslování objektů této knihovny je rozhraní `renderer` a od něj odvozené třídy. Toto rozhraní definuje obecné metody pro vykreslení libovolného prvku rozhraní, které jsou

[1] Tedy při zarovnávání do řady je všem potomkům nastavena stejná výška, při zarovnání do sloupce stejná šířka.

pak implementovány dle potřeb dané odvozené třídy.

Kromě obecných vykreslovacích metod je třeba zmínit parametr `origin`. Tento parametr, společný všem implementacím rozhraní `renderer`, určuje absolutní bod na ploše aplikace, který slouží jako počátek souřadnicového systému pro všechna volání vykreslovacích funkcí. Jeho účelem je usnadnit převod relativních souřadnic, používaných jednotlivými prvky, na absolutní souřadnice na ploše aplikace. Protože všechny prvky ve stejné „generaci“ stromu mají stejný počátek, nastavením tohoto parametru při změně „generace“ odpadá nutnost převádět souřadnice při každém volání vykreslovacích metod. Také je díky tomu možné optimalizovat výpočet absolutní pozice pomocí prostředků nabízených jednotlivými implementacemi.

Poskytnuté vykreslovací metody se jmenují `color_rect`, `texture_rect` a `string_rect`. Jak jejich název napovídá, tyto metody slouží k vykreslení jednobarevného obdélníku, obrazových dat nebo textového řetězce na specifikovanou pozici. Tyto metody slouží jsou primárně určeny pro použití jednotlivými prvky rozhraní, které takto mohou definovat způsob svého vykreslování.

Samotný proces vykreslování pak probíhá zavoláním metody `render` třídy `renderer` nad kořenem stromu prvků. Tato metoda pak s pomocí iterátoru nad tímto stromem navštíví postupně všechny prvky ve stromu a zavolá jejich metodu `render`. Přetížením této metody a používáním výše zmíněných vykreslovaných funkcí definují jednotlivé prvky způsob svého vykreslování.

Kromě vykreslovacích metod a algoritmu pro vykreslení stromu definuje rozhraní `renderer` ještě dvě „přípravné“ metody – `paint_setup` a `paint_cleanup`. Tento pár metod je automaticky volán metodou `render` na úplném začátku, respektive konci renderování stromu, a poskytuje tak jednotlivým implementacím možnost nastavení všech nutných parametrů pro správné vykreslení a poté jejich návratu do stavu před vykreslováním. Díky tomu je možné, aby bylo pro vykreslování možné použít i nastavení, která by ovlivnila i zbytek aplikace.

V aktuálním stavu knihovny je rozhraní `renderer` implementováno za použití specifikace OpenGL. Tato specifikace se zabývá primárně komunikací s grafickou kartou a umožňuje provádět hardwarově urychlené vykreslování. Její součástí je i možnost programovat části procesu rasterizace dat na grafické kartě pomocí tzv. *shaderů*. Díky tomu je možné přesunout výpočetně náročné operace na dedikovaný hardware a významně je tak urychlit. Poskytnutá implementace má v sobě zabudované základní shadery, díky kterým prakticky veškeré výpočty probíhají přímo na grafické kartě. Implementace vykreslovacích metod pak spočívá v nastavení všech souvisejících parametrů, nahrání dat na grafickou kartu a spuštění samotného vykreslování.

Tyto zabudované shadery však nejsou jediné použitelné. OpenGL implementace rozhraní `render` totiž poskytuje i metody pro nahrání vlastních implementací těchto shaderů. Pokud tedy uživatel dodrží rozhraní používané shadery zabudovanými, nic mu nebrání v implementaci různých speciálních efektů pro jím vytvářené grafické uživatelské rozhraní.

Pro implementaci na PC je použita specifikace OpenGL ve verzi 3. Na mobilních zařízeních je však k dispozici pouze omezená verze OpenGL, označená jako OpenGL ES, a ve většině případů pouze ve starší verzi 2. Také jazyk pro definici shaderů se v této verzi lehce odlišuje od verze dostupné v „plném"OpenGL. Implementace `rendereru` pro PC a mobilní zařízení se tedy musí alespoň trochu lišit. Částečně je tento problém řešen vhodným výběrem OpenGL funkcí, které jsou obsaženy v obou použitých verzích specifikace. V případech, kde tento přístup nebyl možný, bylo použito kondičního překladu – během překladu je detekována cílová platforma a s pomocí definic vhodných maker a podmínek preprocesoru je vybrána varianta kódu použitelná na dané platformě.

Co se týče implementace ukládaní obrazových dat, obecné rozhraní je definováno třídou `texture`. Toto rozhraní poskytuje metody pro načtení dat z paměti aplikace a jejich zpřístupnění obecnými metodami. Díky tomu dokáže využít data již v paměti existující a nenutí uživatele načítat je do paměti dvakrát. OpenGL implementace tohoto rozhraní je víceméně prostým zapouzdřením příslušných OpenGL funkcí pro práci s dvourozměrnými texturami. Protože ale práce s obrazovými daty může být náročná na paměť, byl v této oblasti původní návrh rozšířen o koncept „pohledu“ na texturu.

Práce s texturami je tedy implementována pomocí dvou tříd: `texture_storage` a `texture_slice`. Třída `texture_storage` je opravdu zapouzdřením OpenGL metod a jejich zpřístupněním pod obecným programovým rozhraním. Naproti tomu objekt třídy `texture_slice` se vždy vztahuje k nějakému již existujícímu objektu `texture_storage` a specifikuje obdélníkový výřez v uložených obrazových datech. Při vykreslování textury pak volá příslušné OpenGL funkce s upravenými parametry, takže je opravdu vykreslen jen požadovaný obdélník. A protože obě třídy implementují stejné programové rozhraní, lze je libovolně zaměňovat. Je tak možné použít jediný obrázek s grafikou pro všechny části rozhraní, který je pak v aplikaci „rozřezán“ a jeho části používány dle potřeby, což značně zjednodušuje správu grafických zdrojů.

Vykreslování textu

Implementace vykreslování textu nad knihovnou FreeType je implementována zejména pomocí dvou tříd: `font` a `string`. Třída `font` zajišťuje načtení a zpřístupnění fontu ve formátu TrueType nebo OpenType. Tento font je možné načíst jak ze souboru na disku, tak z paměti aplikace. Druhý přístup opět umožňuje využít již existujících dat bez nutnosti jejich znovunačtení z disku. Tento font je pak využíván třídou `string` pro samotnou vizualizaci textového řetězce.

Vykreslení textového řetězce probíhá ve dvou fázích. První z nich se provádí při předání nového řetězce ke zpracování. V tuto chvíli se postupně načtou všechny potřebné znaky ve formátu knihovny FreeType, a společně s doplňujícími informacemi[2] se v tomto formátu uchovají v daném objektu `string`. Ještě však nedojde k samotné rasterizaci těchto znaků, protože není znám konkrétní formát textury, kterou by rasterizace znaků měla vytvořit.

Ve druhé fázi probíhá samotné rasterizování a generování textury s grafickou reprezentací daného textu. V případě požadavku na texturu v daném formátu (tedy například při požadavku na vykreslení textu do `opengl::texture_storage`) se nejdříve ověří, zda již tato textura nebyla vytvořena, a pokud ano, použije se starší verze. V opačném případě je vytvořena dostatečně velká textura požadovaného formátu, do které jsou následně na svoji pozici vykresleny jednotlivé znaky s použitím dříve uložených doplňujících informací.

Účelem tohoto dvoufázového vykreslování je rozložení zátěže. Úkony nezávislé na konkrétním formátu textury jsou provedeny ihned po specifikaci textu, a při samotném vykreslování pak již není třeba je znovu provádět. Také v případě paralelního používání více formátů textury je při každém vykreslení prováděna jen nezbytná část práce. Ukládání dříve vytvořené textury pak také šetří čas při opětovném vykreslování – pokud se nezměnil text, není třeba znovu rasterizovat.

[2] Doplňující informací může být například výsledná pozice znaku se započítáním kerningu.

4.4 Komunikace se zbytkem aplikace

Pro komunikaci se zbytkem aplikace je důležitý zejména mechanismus zpracovávání událostí. Protože má být knihovna použitelná s různými okenními manažery, jsou poskytnuty struktury představující jednotlivé typy objektů. O překlad mezi událostmi okenního manažeru a vnitřními strukturami událostí se stará implementace rozhraní `base_interface`. Poskytnutá implementace tohoto rozhraní je postavena nad knihovnou SDL2 a je obsažena ve stejnojmenné třídě `sdl2`.

Toto komunikační rozhraní je v principu vrstvou, která uživateli knihovny usnadňuje práci s více různými stromy prvků. Základem pro tuto funkčnost jsou metody `attach` a `detach`, které slouží k označení stromů prvků, kterým má být zprostředkována komunikace se zbytkem aplikace (respektive kterým již zprostředkována být nemá).

Překlad událostí zajišťuje metoda `handle_event`. Tato metoda není přímo specifikována v programovém rozhraní `base_interface`, protože neexistuje žádný obecný typ události, kterým by bylo možné popsat vstupní parametr. Jde tedy spíše o konvenci, kterou by měly odvozené třídy dodržovat. Tato metoda má za úkol identifikovat podporované typy událostí, převést je na vnitřní reprezentaci a oznámit ji všem připojeným stromům. Její návratová hodnota pak identifikuje, zda byla událost některým připojeným stromem zpracována. Uživatel tak má možnost jednoduše zaslat událost všem připojeným prvkům a dále pokračovat v závislosti na tom, zda byla či nebyla zpracována.

Druhou pomocnou metodou je metoda `render`. Jejím úkolem je použít renderer, připojený při vytváření komunikační vrstvy, a s jeho pomocí vykreslit všechny připojené prvky. Opět tedy jde hlavně o usnadnění práce programátora, pracujícího s knihovnou – díky této metodě má možnost vykreslit celé rozhraní jediným voláním funkce.

Kapitola 5

Závěr

V této práci jsem se zabýval návrhem a implementací věstavitelné knihovny pro tvorbu grafických uživatelských rozhraní využívající grafickou knihovnu SDL2. Byla navržena struktura obecné knihovny pro tvorbu uživatelských rozhraní, společně se základní sadou ovládacích prvků. Navržená knihovna pak byla implementována právě s použitím knihovny SDL2 a specifikace OpenGL.

Zdrojové kódy knihovny jsou dostupné na přiloženém CD. Součástí těchto kódů jsou i instrukce pro sestavení knihovny programem GNU Make a pro překlad této knihovny pro operační systém Android s použitím sady nástrojů Android NDK.

Cílem této práce bylo seznámit se s platformou SDL2 a rozhraním OpenGL. Obě tyto složky tvoří základní kameny implementace této knihovny. Konkrétní použití zmíněných technologií je popsáno v kapitole 4.

Další částí bylo vytvoření přehledu existujících GUI knihoven určených pro hry, a zhodnocení jejich přínosů a nedostatků. Srovnávané knihovny a technologie jsou popsány v kapitole 2.2, a u každé z nich jsou uvedeny jednotlivé přínosy a potencionální nedostatky.

Návrhem architektury nové GUI knihovny se zabývá kapitola 3. Ačkoliv mělo jít primárně o knihovnu postavenou na platformě SDL2, návrh počítá i s rozšířením knihovny o podporu dalších podobných platforem, a proto je tento návrh koncipován o něco obecněji.

Dalším cílem byla implementace navržené knihovny. Tato implementace se nachází na přiloženém CD, a jejím popisem se zabývá celá kapitola 4.

Zdrojové kódy demonstračních aplikací se také nacházejí na přiloženém CD. Pro PC jde o aplikaci pro operační systém GNU/Linux, pro mobilní zařízení jsou k dispozici zdrojové kódy pro operační systém Android.

Hodnocením přínosu vytvořené knihovny oproti již existujícím knihovnám se věnují následující odstavce. Oproti velkým knihovnám, jako je CEGUI, má implementovaná knihovna východu zejména malé velikosti, a možnosti práce s existujícími daty v paměti programu, bez nutnosti použít pro jejich získání systém specifický pro tuto knihovnu. Výhodou proti podobným knihovnám jako je turbobadger nebo GWEN je kompletní dokumentace veškeré implementované funkčnosti pomocí speciálních komentářů a instrukcí pro dokumentační systém Doxygen. Tuto dokumentaci je možné vygenerovat přímo ze zdrojových kódů jediným příkazem. Oproti knihovně GUIchan je hlavní výhodou využití moderních technologií.

Osobně tuto práci hodnotím jako úspěšnou v oblasti, kterou se zabývala. Všechny cíle se podařilo splnit. Výsledná knihovna je ovšem stále spíše funkčním prototypem než kompletním řešením, a je třeba ji doplnit o větší množství chybějících funkcí.

Literatura

[1] ALEXANDRESCU, A.: *Modern C++ Design: Generic Programming and Design Patterns Applied*. Addison Wesley, 2001, ISBN 978-0-13-338761-2.

[2] BOCKLAGE-RYANNEL, J.; THELIN, J.: *Qt5 Cadaques*. [online], Verze 2015-03, [rev. 2015-05-12], [cit. 2015-05-17].
URL http://qmlbook.org/index.html

[3] CEGUI Community: *CEGUI Community Wiki - Crazy Eddie's Gui System for Games (Open Source)*. [Online], [rev. 2014-06-27], [cit. 2015-02-01].
URL http://cegui.org.uk/wiki/FAQ

[4] EZUST, A.; EZUST, P.: *Introduction to Design Patterns in C++ with Qt, 2nd Edition*. Pearson Education, 2011, ISBN 978-0-13-282645-7.

[5] MITCHELL, S.: *SDL Game Development*. Packt Publishing Ltd, 2013, ISBN 978-1-84-969683-8.

[6] SELLERS, G.; WRIGHT, R. S.; HAEMEL, N.: *OpenGL Superbible: Comprehensive Tutorial and Reference, Sixth Edition*. 2014, ISBN 978-0-321-90294-8.

[7] *Simple Directmedia Layer*. [Online], [cit. 2015-05-01].
URL http://wiki.libsdl.org/FrontPage