Práce s texty v programech pro snadnou jazykovou lokalizaci

Bc. Radim Vymětal

Diplomová práce
2008

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2007/2008

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Radim VYMĚTAL**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Práce s texty v programech pro snadnou jazykovou lokalizaci**

Zásady pro vypracování:

1. **Vytvořte literární rešerši na zadané téma.**
2. **Seznamte se s některými programy, které existují ve více jazykových lokalizacích a proveďte rozbor způsobu této implementace.**
3. **Seznamte se s mezinárodním kódováním UNICODE, chratkerizujte jej a proveďte analýzu jeho implementace v jazyku C/C++.**
4. **Vyberte vhodný způsob implementace vícejazykové aplikace.**
5. **Navrhněte vhodnou vícejazykovou aplikaci, do které implementujete zvolený způsob implementace. Doplňte vhodným popisem pro snadné použití i v jiných aplikacích.**
6. **Tuto aplikaci naprogramujte a odladťe. Pro její tvorbu použijte programovací jazyk C/C++ a knihovnu WxWidgest.**

Rozsah práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **MICHAEL, Morrison. Naučte se programovat počítačové hry za 24 hodin. 1. vyd. Brno : Computer Press, 2004. 421 s. ISBN 80-251-0371-4.**
2. **PETZOLD, Charles. Programování ve Windows. 1. vyd. Praha : Computer Press, 1999. 1216 s. ISBN 80-7226-206-8.**
3. **LIBERTY, Jesse. Naučte se C++ za 21 dní. Brno : Computer Press, 2007. 796 s. ISBN 978-80-251-1583-1.**
4. **WROBLEWSKI, Piotr. Algoritmy - datové struktury a programovací techniky. 1. vyd. Brno : Computer Press, 2004. 351 s. ISBN 80-251-0343-9.**
5. **BLIŽŇÁK, Michal. Systémové programování. Zlín : UTB ve Zlíně, 2005.**
6. **ROŽÁNEK, Filip, GERALT. Jak překládat? [online]. 2001. Dostupný z WWW: http://cestiny.idnes.cz/prekladani/.**
7. **CHALUPA, Radek. Pár slov o UNICODE [online]. 2002. Dostupný z WWW: http://www.builder.cz/art/cpp/winapi5.html.**

Vedoucí diplomové práce: **Ing. Pavel Pokorný, Ph.D.**
Ústav aplikované informatiky

Datum zadání diplomové práce: **20. února 2008**

Termín odevzdání diplomové práce: **19. května 2008**

Ve Zlíně dne 20. února 2008

UNIVERZITA TOMÁŠE BATI VE ZLÍNĚ
Fakulta aplikované informatiky
L.S.

prof. Ing. Vladimír Vašek, CSc.
děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Tato práce se zabývá problémem zobrazování textů v programech napsaných v jiné znakové sadě než je znaková sada programu. Dále pak uchovávání textů mimo zdrojový kód programu a možnosti jeho přeložení do jiného světového jazyka a softwarovou knihovnou pro vytváření multiplatformních aplikací wxWidgets.

Klíčová slova: ASCII, wxWidgets, unicode, utf, jazykové lokalizace, XML

ABSTRACT

My thesis deals with problems connected with displaying texts in programs that use different text character set than the program is written for. Further, the thesis focuses on storing texts outside the program source code, the possibilities of translating into other languages as well as the software library for creating multiplatform applications

Keywords: ASCII, wxWidgets, unicode, utf, language localizations, XML

Rád bych touto cestou poděkoval vedoucímu diplomové práce Ing. Pavlu Pokornému, Ph.D. za odborné vedení, připomínky a pomoc v průběhu řešení této práce. Děkuji také všem konzultantům za jejich odbornou pomoc. Dále pak svým rodičům, kteří mi umožnili studovat a po celou dobu mého studia mě v něm plně podporovali. Také bych rád poděkoval své přítelkyni, která mi byla oporou po celou dobu studia a psaní této práce.

Prohlašuji, že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků, je-li to uvolněno na základě licenční smlouvy, budu uveden jako spoluautor.

Ve Zlíně …….

Podpis diplomanta

OBSAH

ÚVOD

Počítače od svého vzniku vždy komunikovali s uživateli pomocí textů. Nejprve je uživateli vytiskli na papír, později je vypisovali přímo na obrazovku a požadovali aby jim uživatel příslušnou formou odpověděl co chce aby počítač udělal, případně počítač sdělil uživateli co chce aby udělal uživatel.

Vývoj počítačů se nikdy nezastavil ba naopak dělal přímo mílové kroky. Dnes už s námi dokáží počítače komunikovat nejrůznějšími způsoby. Mohou nás na něco upozornit různými zvuky, mohou nám něco říct a dokáží i porozumět když jim něco řekneme my. Dokáží komunikovat i mezi sebou a s jinými přístroji. Nejvíce používanou formou sdělovaní informací jsou však pořád texty, které mohou být vytištěny na papír, vypsány přímo na obrazovce monitoru nebo nám je počítač může zaslat i pomocí textové zprávy na mobil, když nejsme v jeho blízkosti.

Čím více se počítače vyvíjejí tím více se i používají. A čím více se používají tím více funkcí nám nabízejí programy, které nás buď baví, slouží nám k práci či komunikaci. To vše má za následek větší a větší obsah textu v programech. Tyto texty jsou však většinou psány jazykem programátora. Vytvořené programy se pak z velké části používají v nejrůznějších koutech světa a na nejrůznějších softwarových a hardwarových platformách. Tomu se dnes musejí přizpůsobovat i programy. Musí dokázat komunikovat s uživateli nejrůznějším světovým jazykem a běžet na různých platformách. A jakým způsobem se toho dá dosáhnout se zabývá tato práce.

V teoretické části této práce jsou popsány způsoby ukládání textů programu a jejích výhody a nevýhody, multiplatformní knihovna wxWidgets s jejíž pomocí se dá vytvořit program běžící na nejrůznějších softwarových platformách, značkovací jazyk XML, který má mnoho způsobů využití a znaková sada Unicode, pomocí které se napsané texty zobrazí správně na jakémkoli počítači v jakémkoliv koutu světa.

V praktické části je popsán program, který byl vytvořen v programovacím jazyce C++ s pomocí multiplatformní softwarové knihovny wxWidgets, využívající znakovou sadu Unicode a formát XML pro uchovávání textů mimo zdrojový kód programu.

I. TEORETICKÁ ČÁST

1 ZNAKOVÁ SADA ASCII

ASCII je anglická zkratka pro *American Standard Code for Information Interchange*, tedy americký standardní kód pro výměnu informací. V podstatě je to tabulka (Tab. 1 a Tab. 2), která přiřazuje znakům čísla 0 až 255. Kód ASCII je podle původní definice sedmibitový, obsahuje tedy 128 platných znaků. Pro potřeby dalších jazyků a pro rozšíření znakové sady se používají osmibitová rozšíření ASCII kódu, která obsahují dalších 128 kódů. Takto rozšířený kód je přesto příliš malý na to, aby pojal třeba jen evropské národní abecedy [17].

1.1 Přednosti ACII

Existuje hodně dobrých věcí, které se o ASCII dají říct. Všech 25 písmen jde za sebou, velká písmena mohou být převedena na malá písmena a zpět posunem o jeden bit a kódy pro všech 10 číslic se dají snadno odvodit z jejich hodnot. ASCII je velmi spolehlivým a hlavně rozšířeným standardem jako žádná jiná norma. Má zastoupení v klávesnicích, videokartách, v systémovém hardware, tiskárnách, souborech s písmy, operačních systémech a Internetu [12].

1.2 Nevýhody ASCII

Samotný problém standardu ASCII je už v jeho názvu. Je opravdu Americkým standardem což není právě to nejlepší pro ostatní země a to ani pro ty kde se mluví anglicky. Kde má totiž například Angličan hledat znak pro libru £, která se v tabulce ASCII nevyskytuje? Země jako Japonsko, Korea a Čína by v ní své ideogramové znaky hledali úplně marně. Hodnota 255 znaků, které ASCII obsahuje je na pokrytí znaků všech světových jazyků příliš málo. Proto bylo vytvořeno mnoho dalších znakových sad pro různé jazyky například windows-1250 pro češtinu a mnoho dalších. Z toho však vzešel další problém a to převod mezi těmito znakovými sadami. Text napsaný v jedné znakové sadě totiž není čitelný v jiné znakové sadě a převod mezi nimi taky není úplně bezchybný. Tohle vše však řeší znaková sada Unicode.

1.3 Tabulka ASCII

Tabulka je rozdělena na dvě poloviny. V první časti tabulky (Tab. 1) je prvních 32 znaků 00h - 1Fh použito pro řídicí kódy a tyto znaky nelze tisknout. V druhé části (Tab. 2) je standardní tabulka ASCII znaků od 20h do 7Fh. Číslice jsou seřazeny podle hodnoty a písmena podle abecedy, takže je možné, číselný kód ASCII znaků použít pro třídění znaků. V druhé polovině tabulky 80h - FFh jsou národní znaky podle použitého kódování tabulky. Tato horní polovina tabulky zde není uvedena. U každého znaku je uvedeno pořadové číslo v dekadické a hexadecimální formě.

Tab. 1. Netisknutelné znaky ASCII

kód	jméno	význam znaku	kód	jméno	význam znaku
00h	**NUL**	end string	10h	**DLE**	data line escape
01h	**SOH**	start of heading	11h	**DC1**	dev ctrl 1
02h	**STX**	start of text	12h	**DC2**	device ctrl 2
03h	**ETX**	end of text	13h	**DC3**	dev ctrl 3
04h	**EOT**	end of transmission	14h	**DC4**	device ctrl 4
05h	**ENQ**	enquiry	15h	**NAK**	negative acknowledge
06h	**ACK**	acknowledge	16h	**SYN**	synchronous idle
07h	**BEL**	bell - zvonek	17h	**ETB**	end transmit block
08h	**BS**	backspace	18h	**CAN**	cancel
09h	**HT**	TAB horizontal tab - tabulátor	19h	**EM**	end of medium
0Ah	**LF**	line feed - posun o řádek	1Ah	**SUB**	substitute
0Bh	**VT**	vertical tab - nový řádek	1Bh	**ESC**	escape
0Ch	**FF**	form feed - posun o stránku	1Ch	**FS**	file separator
0Dh	**CR**	carriage return - návrat vozíku	1Dh	**GS**	group separator
0Eh	**SO**	shift out - nový slopec	1Eh	**RS**	record separator
0Fh	**SI**	shift in	1Fh	**US**	unit separator

Tab. 2. Standardní znaky ASCII do hodnoty 128

HEX	znaky	HEX	znaky	HEX	Znaky	HEX	Znaky	HEX	Znaky	HEX	znaky
20h		30h	**0**	40h	@	50h	**P**	60h	`	70h	**p**
21h	**!**	31h	**1**	41h	**A**	51h	**Q**	61h	**a**	71h	**q**
22h	**"**	32h	**2**	42h	**B**	52h	**R**	62h	**b**	72h	**r**
23h	**#**	33h	**3**	43h	**C**	53h	**S**	63h	**c**	73h	**s**
24h	**$**	34h	**4**	44h	**D**	54h	**T**	64h	**d**	74h	**t**
25h	**%**	35h	**5**	45h	**E**	55h	**U**	65h	**e**	75h	**u**
26h	**&**	36h	**6**	46h	**F**	56h	**V**	66h	**f**	76h	**v**
27h	**'**	37h	**7**	47h	**G**	57h	**W**	67h	**g**	77h	**w**
28h	**(**	38h	**8**	48h	**H**	58h	**X**	68h	**h**	78h	**x**
29h	**)**	39h	**9**	49h	**I**	59h	**Y**	69h	**i**	79h	**y**
2Ah	*	3Ah	**:**	4Ah	**J**	5Ah	**Z**	6Ah	**j**	7Ah	**z**
2Bh	**+**	3Bh	**;**	4Bh	**K**	5Bh	**[**	6Bh	**k**	7Bh	**{**
2Ch	**,**	3Ch	**<**	4Ch	**L**	5Ch	****	6Ch	**l**	7Ch	**|**
2Dh	**-**	3Dh	**=**	4Dh	**M**	5Dh	**]**	6Dh	**m**	7Dh	**}**
2Eh	**.**	3Eh	**>**	4Eh	**N**	5Eh	**^**	6Eh	**n**	7Eh	**~**
2Fh	**/**	3Fh	**?**	4Fh	**O**	5Fh	**_**	6Fh	**o**	7Fh	

2 UNICODE

Počítače, ze své podstaty, pracují pouze s čísly. Písmena a další znaky ukládají tak, že každému z nich přiřadí číslo. Před vznikem Unicode existovaly stovky rozdílných kódovacích systémů pro přiřazování těchto čísel (ASCII, PC Latin 2, Windows 1250, ISO Latin 2). Žádné z těchto kódování nemohlo obsahovat dostatek znaků: například Evropská unie sama potřebuje několik různých kódování, aby pokryla všechny své jazyky. Dokonce i pro jeden jediný jazyk, jako je angličtina, nevyhovovalo žádné kódování pro všechny písmena, interpunkci a běžně používané technické symboly [16].

Tyto kódovací systémy také byly v konfliktu jeden s druhým. To znamená, že dvě kódování mohou používat stejné číslo pro dva různé znaky, nebo používat různá čísla pro stejný znak. Jakýkoli počítač (zvláště servery) musí podporovat mnoho různých kódování; přesto, kdykoli jsou data předávána mezi různými kódováními nebo platformami, hrozí, že tato data budou poškozena [16].

2.1 Co je Unicode?

Je to univerzální kódování, které pracuje tak, že každému znaku přiřazuje jedinečné číslo, nezávisle na platformě, nezávisle na programu, nezávisle na jazyku. Výsledkem je v podstatě tabulka znaků, kde je každému znaku přiřazena specifická hodnota, která je určena jen pro konkrétní znak. Unicode Standard byl přijat takovými průmyslovými vůdci, jako jsou Apple, HP, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys a mnoha dalšími. Unicode je vyžadován moderními standardy, jako jsou XML, PHP, Java, ECMAScript (JavaScript), LDAP, CORBA 3.0, WML atd. a je oficiální formou implementace ISO/IEC 10646. Je podporován v mnoha operačních systémech, všech moderních prohlížečích a mnoha dalších produktech. To, že se objevil Unicode Standard a dostupnost nástrojů, které jej podporují, patří mezi nejvýznamnější nedávné trendy v globální technologii softwaru [16].

Začlenění Unicode do klient-server nebo vícevrstvých aplikací a webových stránek nabízí významné ušetření nákladů oproti dřívějším znakovým sadám. Unicode umožňuje, aby jediný softwarový produkt nebo jediná webová stránka byla zaměřena na mnoho platforem, jazyků a zemí beze změn návrhu. To dovoluje přenášet data bez porušení přes

mnoho různých systémů a aplikací [16]. Odpadá složité a nesprávně převádění textu z jednoho druhu kódování do jiného. Díky Unicode lze zobrazit všechny znaky zároveň v jednom textu, lze tedy kombinovat například češtinu, ruštinu a němčinu.

2.2 Znaky Unicode

Základní problém spočívá v tom, že každý světový jazyk nelze reprezentovat pomocí 256 8bitových kódů. Programátoři s problémy toho typu mají zkušenosti. Když nestačí 8bitové hodnoty, zkusí se širší hodnoty například 16bitové. A to je myšlenka, která stojí za vytvořením Unicode. Namísto zmatku s manipulováním 256 znaků nebo dvoubytovou znakovou sadou, která má některé kódy jednobytové, Unicode je jednotný 16bitový systém, který tedy umožňuje vyjádření 65 536 (2^{16}) znaků [12]. To je dostatečné číslo pro pokrytí všech znaků a ideogramů psaných ve všech jazycích světa a pro speciální vědecké a matematické symboly, měnové jednotky, kombinované znaky a podobně.

Každý znak má jednoznačný číselný kód a svůj název jak znázorňuje tabulka (Tab. 3). Binární podobu řeší až konkrétní kódování. Prvních 128 znaků Unicode jsou ASCII, v 16bitovém kódování jsou to hodnoty 0x0000 až 0x007F. Dalších 128 znaků v 16bitovém kódování tedy hodnoty 0x0080 až 0x00FF jsou ISO 8859-1 rozšíření k ASCII. Různé bloky kódů uvnitř Unicode jsou podobným způsobem založeny na stávajících standardech. Například řecká abeceda používá kódy od 0x0370 po 0x03FF a ideogramy čínštiny, japonštiny a korejštiny označované jednotně jako CJK zabírají kódy od 0x3000 do 0x9FFF. Nejlepší na Unicode je to, že je to jedna znaková sada, není žádná dvoujazyčnost a tak se pomocí ní i mezi jinými znakovými sadami snadno převádí [12].

Tab. 3. Tabulka českých znaků Unicode

znak	HTML	dec	hex	UTF-8	znak	HTML	dec	hex	UTF-8
Á	Á	Á	Á	%C3%81	**á**	á	á	á	%C3%A1
Č	Č	Č	Č	%C4%8C	**č**	č	č	č	%C4%8D
Ď	Ď	Ď	Ď	%C4%8E	**ď**	ď	ď	ď	%C4%8F
É	É	É	É	%C3%89	**é**	é	é	é	%C3%A9
Ě	Ě	Ě	Ě	%C4%9A	**ě**	ě	ě	ě	%C4%9B
Í	Í	Í	Í	%C3%8D	**í**	í	í	í	%C3%AD
Ň	Ň	Ň	Ň	%C5%87	**ň**	ň	ň	ň	%C5%88
Ó	Ó	Ó	Ó	%C3%93	**ó**	ó	ó	ó	%C3%B3
Ř	Ř	Ř	Ř	%C5%98	**ř**	ř	ř	ř	%C5%99
Š	Š	Š	Š	%C5%A0	**š**	š	š	š	%C5%A1
Ť	Ť	Ť	Ť	%C5%A4	**ť**	ť	ť	ť	%C5%A5
Ú	Ú	Ú	Ú	%C3%9A	**ú**	ú	ú	ú	%C3%BA
Ů	Ů	Ů	Ů	%C5%AE	**ů**	ů	ů	ů	%C5%AF
Ý	Ý	Ý	Ý	%C3%9D	**ý**	ý	ý	ý	%C3%BD
Ž	Ž	Ž	Ž	%C5%BD	**ž**	ž	ž	ž	%C5%BE

2.3 Co znamená UTF-8 a UTF-16?

Slovo Unicode (a ISO 10646) se používá k označení znakové sady. Přiřazují jednotlivým znakům čísla. Například znak „**í**“ má přiřazeno číslo 237 (Tab. 1). Naproti tomu UTF-8 a UTF-16 jsou různé způsoby zápisu znaku do souboru. Například znak číslo 237 bude pomocí UTF-8 zapsán jako 11000011-10101101, zatímco pomocí UTF-16 bude zaspán jako 11101101-00000000 [19].

2.3.1 UTF-8

Je způsob zápisu znaků Unicode do souboru (přesněji řečeno znaků ISO 10646). ASCII znaky (písmena latinky bez háčků a čárek) jsou reprezentována jedním byte. Všechny ostatní znaky jsou prezentovány více byty. Pokud dokument zapsaný pomocí UTF-8 obsahuje mnoho ASCII znaků, pak je jakž takž čitelný i v editorech nepodporujících Unicode [19].

2.3.2 UTF-16

Je jiný způsob zápisu znaků Unicode. Skoro všechny znaky jsou prezentovány pomocí dvou byte (některé hodnoty mají speciální význam). Tento způsob zápisu je jednodušší než

UTF-8. Pro programy nepodporující Unicode však budou dokumenty zakódované pomocí UTF-16 naprosto nečitelné [19].

2.4 Nevýhody Unicode

Jistě, že Unicode má i nevýhody a to větší délku textu jelikož jeho hodnoty jsou 16bitové. To také znamená, že zabírají více místa v paměti než například 8bitové ASCII kódování. Text je po převodu z osmibitového kódování do Unicode dvojnásobně dlouhý, zdánlivě bez přidání informační hodnoty. Výsledek zabere víc místa při uložení a také následné zpracování je pomalejší [3].

256x větší znaková sada. Znaky na počítači se zobrazují pomocí fontů, které pro Unicode musí obsahovat 256x víc znaků než pro osmibitové znakové sady. Vzhledem k tomu, že v mnoha jazycích se použije jen nepatrný zlomek celkového množství, znaky navíc zbytečně zabírají místo. Microsoft tento problém řeší tak, že ani Unicode fonty neobsahují všechny znaky, ale pouze ty, které jsou používány v daném prostředí - a v případě potřeby je možno pořídit plné fonty [3].

Nekompatibilita s osmibitovým prostředím. Unicode text může "legálně" obsahovat znaky, které v "normálním", osmibitovém textu obvykle nejsou a zpravidla mají speciální význam - jedná se zejména o binární nulu, kterou Unicode text může obsahovat jako vyšší byte dvoubytového kódu. Nelze tedy použít stávající programový kód pro práci s textem - ten se musí od základu přepsat [3].

Nelze jednoduše zjistit, zda je daný text v Unicode nebo ne. Nelze proto zároveň používat Unicode a nějaké osmibitové kódování. Není tedy možné přejít na Unicode částečně, je nutný globální přechod, aby aplikace nejen dostaly text v Unicode, ale aby hlavně Unicode text i očekávaly. Fakt, že to jde v rámci uzavřeného systému, předvádí názorně firma Sun s Javou, pracující interně v Unicode, a také produkty firmy Microsoft. Podporu Unicode však nelze zajistit obecně a nelze počítat s tím, že mu druhá strana bude vždy rozumět [3].

2.5 Použití Unicode

2.5.1 Databázové systémy

Dnes již většina databázových systému (MySQL, Oracle a jiné.) používá ukládání dat pomocí kódovacího formátu Unicode respektive UTF-8. Ovšem aplikace kterým uložená data v databázových systémech patří nemusí používat právě kódováni Unicode. Dokonce mohou používat pro zápis dat jiné kódování než pro čtení i když se to stává zřídka. Proto se ještě musí nastavovat druh kódování příslušné tabulky v databázi. Například pokud budeme mít dvoujazyčnou webovou stránku a budeme český text zobrazovat pomocí kódování *windows-1250* a Arabský text pomocí *ASMO-708*. Použité kódování budeme muset použít i pro přístup k databázi v *connection stringu* webové stránky. Například pro MySql databázi mohou příkazy vypadat takto:

mysql_query("SET CHARACTER SET cp1250");

mysql_query("SET collation_connection=cp1250_general_ci");

2.5.2 WWW stránky

Při psaní stránek www je situace stejná jak pro HTML, PHP, XML a další druhy jazyků. V podstatě se jen nastavuje typ kódování textu pro uložení do souboru direktivou *charset* v hlavičce www stránky. To tedy znamená, že v případě použití Unicode nenapíšeme do direktivy unicode, ale příslušný druh zápisu do souboru, tedy UTF-8. Direktiva charset bude tedy vypadat takto: *content="text/html; charset = utf-8".*

2.5.3 Tvorba aplikací

V programování se zatím moc Unicode nevyužívá, ale cestička k jeho použití v programovacích jazycích už je připravená. To, že se ovšem unicode moc nevyužívá neznamená, že jej nemůžeme najít vůbec. Každý z nás denně pracuje z takovou aplikací aniž by jsi to uvědomoval, samotný operační systém je totiž napsán pomocí unicode. První verze u Windows, která podporovala Unicode byla verze Windows 98, která jej podporovala jen částečně a Windows NT, která jej podporovala plně. Proč psát aplikace

s využítím Unicode nejlépe právě ukazuje samotný operační systém. Pokud by nebyl psán pomocí Unicode musel by mít daleko více verzí. Ano má i teď několik verzí, ale jen co se jazyka týče. To ovšem neznamená, že má i několik druhů zdrojových kódů jak tomu muselo být dříve. Dřívější verze musely mít zdrojové kódy napsány i pro jiné země, kterým kódování pomocí 256 znaků nestačilo. Ideogramy Číny, Japonska a Koreje čítají něco okolo 21 000 znaků a s tím si ASCII rozhodně poradit nemohlo. Proto musely být napsány verze Windows speciálně pro tyto země. Používání Unicode to vše však mění [12].

2.6 Implementace Unicode v jazyce C/C++

Zde je předpokládáno, že čtenář má základní programovací znalosti v jazyce C/C++. Fakt, že *char* má stejnou velikost jako jeden bajt je jedna z mála jistot v tomto životě. Proto se někteří programátoři obávají zda vůbec ANSI C podporuje znakové sady, které vyžadují více jak jeden bajt pro znak. Ano, podporuje a dokonce mohou existovat společně s normálními dobře známými znaky, říká se jim *široké znaky*. Široké znaky nemusí být nutně kódy Unicode. Je to jen jedna z možností kódování znaků [12].

2.6.1 Široké znaky WCHAR

Nic co se Unicode nebo širokých znaků týká nemůže změnit význam datového typu *char* v jazyce C. Široké znaky jsou v jazyce C založeny na datovém typu *wchar_t*, který je deklarován v několika hlavičkových souborech včetně WCHAR.H. Každý znak datového typu zabírá 2 byty. Deklarace je:

Typedef unsigned short wchar_t;

Deklarace proměnné pro jeden znak se píše takto: *wchar_t c = "A";* nebo

wchar_t c = L"A"; velké písmeno *L* (*long*) je napsáno těsně před uvozovkou. To překladači říká, že tento řetězec má být uložen jako široké znaky.

Deklarace ukazatele na řetězec širokých znaků: *wchar_t * p = L"Hello!";*

Pole širokých znaků se pak deklaruje obdobně: *static wchar_t a[] = L"Hello!";*

Pro zjišťování délky řetězce se v případě datového typu *char* volá funkce *strlen()*, v případě širokých znaků a tedy datového typu *wchar_t* se pro zjištění délky řetězce musí používat funkce *wcslen()* [12].

2.6.2 Zjednodušení pomocí TCHAR

Samozřejmě zápis pomocí wchar je zdlouhavější, nesmí se zapomínat na L před uvozovkami a navíc zabere dvakrát tolik místa v programu. Knihovní funkce v run-time mají odlišné názvy a i znaky se deklarují odlišně. Jedním z řešení může být hlavičkový soubor TCHAR.H. Ne však součástí standardní normy ANCI C a tak každá definice funkce má v tomto souboru na začátku podtržítko. V podstatě se jedná o to, že TCHAR.H obsahuje alternativní názvy pro normální funkce run-time. To znamená, že pokud je definován identifikátor s názvem _UNICODE a hlavičkový soubor TCHAR.H je funkce *_tcslen* deklarována jako *scelen*, datový typ *char* jako *wchar*, zároveň je deklarováno i makro _T (_TEXT) a způsobuje, že písmeno L je automaticky připojeno k parametru makra například *_TEXT ("Hello!")*. V případě, že je identifikátor _UNICODE deklarován, bude řetězec chápán jako skupina širokých znaků, v opačném případě bude chápan jako skupina 8bitových znaků [12].

2.6.3 Hlavičkový soubor WINDOWS

Hlavičkový soubor WINDOWS.H obsahuje řadu dalších hlavičkových souborů včetně WINDEF.H, ve kterém je většina základních typů používaných ve Windows a navíc obsahuje hlavičkový soubor WINNT.H a ten se stará o základní podporu Unicode. WINNT.H navíc definuje TCHAR jako generický datový typ. Identifikátor UNICODE je bez podtržítka. Hlavičkové soubory WINNT.H a WCHAR.H jsou chráněny proti opětovné definici datového typu TCHAR. Proto v případě používání i jiných hlavičkových souborů by se měl hlavičkový soubor WINDOWS.H umisťovat před ostatní [12].

V následujícím výpisu je vidět proškrtaný hlavičkový soubor WINNT.H, součást hlavičkového souboru WINDOWS.H:

```
typedef char CHAR;

typedef wchar_t WCHAR;

#ifdef UNICODE

typedef WCHAR TCHAR, *PTCHAR;

#else

typedef char TCHAR, *PTCHAR;

#endif
```

Pokud jde o použití konstantních řetězců, k jejich správné interpretaci podle nastavení Unicode slouží makro TEXT

```
TEXT(

        LPTSTR string  // ANSI or Unicode string

);
```

Toto makro identifikuje text jako Unicode, pokud je definována hodnota UNICODE, v opačném případě jako ANSI řetězec. Máli tedy být kód bez problémů přeložitelný i v případě definovaného Unicode, musí se toto makro důsledně používat. Příkladem je definice

#define _AppName TEXT("Učíme se WinAPI") [6].

3 MULTIPLATFORMNÍ SOTWAROVÁ KNIHOVNA WXWIDGETS

WxWidgets dříve zvaná jako wxWindows je soubor softwarových knihoven pro programovací jazyk C++, ale její použití je možné i v jiných běžných programovacích jazycích jako například Python (wxPython), C#, Perl (wxPerl), Ruby (wxRuby), Java a také JavaScript (wxJS). Je to knihovna umožňující multiplatformní programování.

3.1 Vývoj wxWidgets

Vývoj této knihovny začal Julian Smart v roce 1992, který měl za úkol vytvořit aplikaci běžící na platformě Windows a X-Windows. Vytvořil tehdy knihovnu wxWindows (w jako Windows, x jako X-Windows) s podporou technologií XView a MFC. Později bylo upuštěno od přímého využívání knihovny MFC a tím bylo umožněno knihovnu wxWidgets šířit jako open source [2].

3.2 Současné možnosti wxWidgets

V současné době knihovna wxWidgets umožňuje kompilaci programů na několika různých softwarových platformách a překladačích s minimálními změnami zdrojového kódu vytvářené aplikace. Pro každou podporovanou platformu existuje jedna verze této knihovny. Knihovna obsahuje základní API funkce pro tvorbu grafického rozhraní aplikace GUI , ale i další obecně dostupné technologie jako souborové operace, řízení procesů a vláken, časovače a různé třídy pro síťové operace, XML, 2D a 3D grafiku a další [2].

3.3 Přechod z MFC na wxWidgets

Knihovna wxWidgets je natolik podobná knihovně MFC pro platformu Windows, že pro programátora v MFC je přechod na wxWidgets jen otázkou páru dní [2]. Vlastně jde jen o to aby se programátor naučil nové názvy již běžně používaných tříd a jejich proměnných v MFC a dále se naučil pár nových tříd sloužících k operacím, které MFC neumožňuje. S tím mu velice dobře pomůže obsáhlá nápověda knihovny wxWidgets. Nevětší rozdíl oproti API je pak v tom, že wxWidgets si stará o obsluhu smyčky správ sama. Programátor se pak soustředí převážně na své funkce.

4 JAZYKOVÉ LOKALIZACE PROGRAMŮ

Slovem lokalizace nejčastěji rozumíme překlad hry nebo programu do jiného jazyka, ale v obecné rovině jde o přizpůsobení konkrétním kulturním zvyklostem v dané zemi. Český trh není z největších, a každý aspoň trochu ambiciózní projekt by měl disponovat minimálně anglickou verzí, aby bylo možno jej prezentovat za hranicemi naší vlasti. Konec konců, i pravidla populární soutěže Becher Game [5] vyžadují, aby soutěžní hry bylo možné hrát jak v češtině, tak v angličtině. Aby mohla být hra snadno lokalizována, měla by být na lokalizaci připravena [8].

4.1 Ukládání textů v programech

Kde se tedy nejčastěji ukládají texty v programech? Tato otázka ani není tak důležitá jako otázka kam texty při psaní programu ukládat. Texty můžete v podstatě ukládat kam vás to jen napadne, ale měli by jste myslet na to zda budete chtít texty později přeložit i do jiného jazyka nebo to za vás bude dělat dokonce někdo jiný, jak jsou které texty důležité a jestli je tedy bude nutno přeložit. Dnes se v podstatě používá několik nejrůznějších způsobů ukládaní textů od ukládání přímo v programovém kódu až po externí soubory.

4.1.1 Ukládání textů přímo do zdrojového kódu

Tento způsob je asi nejstarší formou ukládání textů v programech a do jisté míry se používá u všech ostatních způsobů. Jde o to, že text se uloží vždy přímo do té části programového kódu kde nebo spíše odkud má být vypsán. Po překompilování programu pak texty v postatě zůstanou uloženy ve výsledném EXE souboru. To je právě ten kámen úrazu. Pokud někdo bude chtít váš program přeložit nebude to mít nějak jednoduché a to ani pokud mu poskytnete zdrojový kód jelikož v něm bude všechny tyto texty pracně hledat což by jste dělali i vy sami. Z tohoto důvodu je efektivnější psát texty jinam. V programovém kódu pak místo textů zůstanou jen indexy odkazující na příslušný text, který se nachází jinde se všemi ostatními. Ostatní způsoby se tedy odvíjejí od tohoto způsobu a jak tedy už bylo řečeno tento způsob se používá i při ukládání do externích souborů apod.

Příklad takovéhoto použití ukládání textu může vypadat například takto:

Franta.Talk("Ahoj Pepo, jak se máš?");
Pepa.Talk("Ale jo, ujde to.");
Franta.Talk("Tak já už musím běžet. ");

Tento způsob ukládání textů používá například program pro přehrávání hudby a videa RealPlayr [13].

4.1.2 Ukládání textů do externích souborů

Příklady k lokalizaci textů se různí, ale v ideálním případě by měly být všechny texty soustředěny pěkně pohromadě v jednom souboru, kterému se říká tabulka řetězců (string table). Ten se pak dá předat překladateli, který jej přeloží bez menších problémů. Aby se toho dalo dosáhnout musí být každý text ve hře vybaven jednoznačným identifikátorem. Program se pak bude interně odkazovat na texty jen pomocí tohoto identifikátoru, a vlastní text bude možné libovolně vyměňovat podle aktuálně zvoleného jazyka [8].

Přiřazení jednoznačných identifikátorů jednotlivým textům je poměrně zdlouhavá nepříjemná věc, jelikož je obtížné je spravovat ručně a přitom se vyhnout duplicitám, na žádný text nezapomenout a podobně. Samozřejmě, záleží na typu hry a potažmo na množství textů. U akční hry, kde se překládá jen "Spustit hru" a "Zabili tě", se udrží tabulku řetězců snadno i ručně, ale u RPG či adventury kde je velké množství textů už to tak snadné není. Nejlepší přístup asi je vyvíjet hru (relativně) bez ohledu na budoucí lokalizaci, a teprve když je hra v dostatečně finální podobě, se pomocí nějakého nástroje z herních souborů vykousnou všechny texty určené k překladu a vyexportují se do tabulky řetězců. Při tomto procesu se zároveň každému textu přiřadí zmíněný identifikátor, a text v původním souboru je nahrazen tímto identifikátorem [8].

K tomuto způsobu ukládaní textů se nejčastěji používají textové soubory s příponou TXT nebo LANG a soubory XML, které mají o trochu jinou strukturu a umožňují použití různých entit, které se mohou výborně hodit při definování jednotek real-time strategii. Všechno jsou to takzvané datové soubory programu. Pro tyto datové soubory je dobré

používat textový formát místo binárního. Příklad použití textového souboru může vypadat například takto:

Programový kód: *Franta.Talk("STRING0001");*
Pepa.Talk("STRING0002");
Franta.Talk("STRING0003");

String table: *STRING0001 Ahoj Pepo, jak se máš?*
STRING0002 Ale jo, ujde to.
STRING0003 Tak já už musím běžet.

Tohoto způsobu ukládání textů využívá například souborový manažer Total Commander [11].

4.1.3 Obrázky

Každého hned asi napadne, že použití obrázků pro ukládání textů asi není zrovna ideální, ale někdy se jejich použití někomu hodí. Obrázky se mohou ukládat jako externí soubory, ale mohou být i uloženy jako zdrojový soubor kdy se všechny použité obrázky zkomprimují a uloží jako jeden výsledný soubor kde jsou všechny obrázky uloženy vedle sebe a pod sebou. Tento soubor je po kompilaci součástí EXE souboru případně je uložen jako externí zdrojový soubor například s příponou RES, RC a jiné. Mohlo by se zdát, že takto uložené text jsou už nepřeložitelné, ale právě naopak. Dá to sice trochu více práce, ale pro lidi zabývající se tvorbou češtiny je to jednoduchá věc. Stačí je vyexportovat na disk pomocí nějakého programu jako je například Resource Hacker a poté je upravit nebo vytvořit nové pomocí grafického editoru a pak je v programu nahradit což udělá ten samý program, který je vyexportoval [14]. Tohle řešení by se však k ukládání textů používat nemělo slouží spíše jen pro nějaký text v ložený v obrázku, u kterého se nepředpokládá, že by se vůbec měnil nebo překládal.

4.1.4 Statické a dynamické knihovny

Knihovny v podstatě poskytují služby pro programy. Většinou se jedná o sbírku procedur, funkcí, datových typů, objektově orientovaných přístupů a podobně. A právě

v procedurách a funkcích se mohou nacházet další texty. Knihovny se používají především u rozsáhlejších projektů a hlavně tam, kde program obsahuje více aplikací, které používají stejné funkce. Rozdíl mezi statickou a dynamickou knihovnou je především ve využití paměti. Statická knihovna se pro každou aplikaci nahrává do paměti zvlášť což znamená, že může být v paměti několikrát. Oproti tomu dynamickou knihovnu je možno sdílet, takže je v paměti načtena jen jednou a aplikace k ní přistupují dle určených pravidel, o které se stará operační systém. Statická knihovna je o něco málo rychlejší než dynamická, ale dynamicky slinkovaný program je výrazně menší a pokud tedy knihovnu využívá více než jeden program ušetří se místo na disku. Knihovny programů jsou stejně jako soubory s obrázky zdrojové části programu. Nejčastěji mají příponu DLL, LIB, ale i jiné.

Tento způsob ukládání textů jazykových verzí využívá například program Daemon Tools [4], vytvářející virtuální mechaniku pro soubory obrazů CD a DVD.

Tohle však není vše. Určitě by se daly najít ještě další typy souborů kde by se daly jazykové texty ukládat. I zdrojů programu je více a jsou tam další texty. Zdrojů (resources) je totiž hned několik typů. Jsou to Dialog (okna), Menu (nabídky), Cursor (kurzory), Strings (hlášení), String Tables (skupiny hlášení), Version Info (informace o verzi programu), Icon (ikony), Bitmap (obrázky), Accelerators (zkratkové klávesy) a občas se vyskytují i Wave (zvuky) a HTML (internetové stránky) [14].

5 ZNAČKOVACÍ JAZYK XML

XML (*eXtensible Markup Language*, česky *rozšiřitelný značkovací jazyk*) je obecný značkovací jazyk, který byl vyvinut a standardizován konsorciem W3C. Umožňuje snadné vytváření konkrétních značkovacích jazyků pro různé účely a široké spektrum různých typů dat [18].

Jazyk je určen především pro výměnu dat mezi aplikacemi a pro publikování dokumentů. Jazyk umožňuje popsat strukturu dokumentu z hlediska věcného obsahu jednotlivých částí, nezabývá se sám o sobě vzhledem dokumentu nebo jeho částí. Prezentace dokumentu (vzhled) se potom definuje připojeným stylem. Další možností je pomocí různých stylů provést transformaci do jiného typu dokumentu, nebo do jiné struktury XML [18].

V současné době se o XML hovoří nejčastěji v souvislosti s Webem a považuje se za nástupce dnes používaného jazyka HTML. Výhoda XML spočívá v tom, že autor stránky může používat vlastní tagy, které dokáží mnohem přesněji označit význam prezentovaných informací. To má velký význam především pro vyhledávání informací. Dnešní Web je informacemi přehlcen a nalézt konkrétní informací je stále obtížnější a mnohdy i nemožné. Tento problém nevyřeší sebelepší prohledávací servery, pokud jim nepomohou autoři stránek, kteří pomocí XML uloží do stránek mnohem více metainformací [7].

Stránky v XML jsou zároveň snazší pro čtení než ty současné v HTML, které obsahují mnoho chyb. To umožní vývoj nových jednoduchých prohlížečů určených zejména pro různá kapesní mobilní zařízení [7].

V dnešní době již není vhodné zasílat dokumenty v nějakém tvaru, který vyžaduje speciální software konkrétní firmy, jako je např. formát DOC, XLS nebo T602. Je používána celá řada operačních a informačních systémů a není záruka, že každý uživatel vlastní příslušný software [18].

Vznikla tak potřeba vytvořit nějaký jednoduchý otevřený formát, který není úzce svázán s nějakou platformou nebo technologií. Tím může být právě XML, který je založen na jednoduchém textu a je zpracovatelný (v případě potřeby) libovolným textovým editorem [18].

Specifikace XML konsorcia W3C je zdarma přístupná všem. Každý tak může bez problémů do svých aplikací implementovat podporu XML. To je velký rozdíl oproti firemním formátům, k nimž není k dispozici žádná dokumentace a navíc se jedná v porovnání s XML o velice složité, často binární, formáty [18].

5.1 Mezinárodní podpora

XML hned od samého počátku myslel na potřeby i jiných jazyků než je angličtina. Jako znaková sada se implicitně používá ISO 10646 (také Unicode). V XML proto můžeme vytvářet dokumenty, které obsahují texty v mnoha jazycích najednou – můžeme přepínat mezi různými jazyky v jednom dokumentu. Současně je přípustné i jiné libovolné kódování (např. windows-1250, iso-8859-2) [18]. Standardně se předpokládá, že dokument bude uložen v kódování UTF-8, které má prvních 128 znaků shodných s ASCII. Pokud v dokumentu použijeme jiné kódování, musíme to uvést v XML deklaraci [7]. Deklarace UTF-8 pak může vypadat takto <?xml version="1.0"?> nebo takto <?xml version="1.0" encoding="utf-8"?>. Deklarace jiného kódování pak třeba takto <?xml version="1.0" encoding="iso-8859-2"?>.

5.2 Hypertextové odkazy

XML stejně jako HTML umožňuje vytváření odkazů v rámci jednoho dokumentu i mezi dokumenty, má však více možností. Je možné vytvářet i vícesměrné odkazy, které spojují více dokumentů dohromady. Tvorba odkazů je popsána ve třech standardech – XLink, XPointer a Xpath [18], které někdy bývají souhrnně označovány jako XLL.

XPath (XML Path Language) je jazyk, který umožňuje adresovat jednotlivé části dokumentu [18].

XPointer (XML Pointer Language). Je rozšířením XPath. Používá k určování jednotlivých částí dokumentu ve stylu: „zajímá mě první odstavec třetí kapitoly“. Není nutné ty části dokumentu, na které chceme odkazovat, explicitně označovat pomocí návěstí jako v HTML [18].

XLink (XML Linking Language) je samotný jazyk pro tvorbu odkazů. Jednotlivé dokumenty se určují pomocí jejich URL adresy, za kterou lze uvést ještě XPointer pro přesnější určení části dokumentu [18].

XLL samozřejmě umožňuje vytváření jednoduchých odkazů, které známe z HTML. Přichází však s velkou novinkou. Odkaz nyní může vést i na tu část stránky, která neobsahuje žádné návěstí. Odkazy v XLL mohou být definovány mnohem flexibilnějším způsobem. Můžeme se například odkázat na třetí odstavec, který následuje po kapitole, jež se jmenuje "Ze života hmyzu". Tímto způsobem můžeme vytvářet odkazy i na ta místa dokumentu, kam autor neumístil žádné návěstí [7].

Další výhodou je možnost vytvářet obousměrné odkazy, které jasně vyjadřují souvislost dvou stránek. Problémem nejsou ani odkazy, kdy jeden odkaz ukazuje najednou na několik dalších zdrojů. Uživatel pak má možnost vybrat si zdroj, který jej nejvíce zajímá [7].

Odkazy lze pomocí XLL ukládat i zcela mimo dokumenty, kterých se týkají. Tímto způsobem se dají vytvářet například anotace ke stránkám, ke kterým nemáme jinak přístup pro zápis. Na svém počítači nebo na firemním serveru můžeme mít databázi odkazů, se kterou bude spolupracovat náš prohlížeč, a budeme pak mít na cizích stránkách přístupné anotace vlastní i kolegů. Není vyloučeno, že se najdou provozovatelé serverů, které budou spravovat veřejné databáze odkazů. Stránky tak půjde snadno doplňovat o další informace, které mohou být užitečné pro všechny uživatele Webu [7].

5.3 Syntaxe XML

Efektivita XML je silně závislá na struktuře, obsahu a integritě. Aby byl dokument považován za správně strukturovaný („well-formed“), musí mít nejméně následující vlastnosti:

- Musí mít právě jeden kořenový (root) element.
- Neprázdné elementy musí být ohraničeny startovací a ukončovací značkou. Prázdné elementy mohou být označeny tagem „prázdný element“.

- Všechny hodnoty atributů musí být uzavřeny v uvozovkách – jednoduchých (') nebo dvojitých ("), ale jednoduchá uvozovka musí být uzavřena jednoduchou a dvojitá dvojitou. Opačný pár uvozovek může být použit uvnitř hodnot.
- Elementy mohou být vnořeny, ale nemohou se překrývat; to znamená, že každy (ne kořenový) element musí být celý obsažen v jiném elementu.
- Jména elementů v XML rozlišují malá a velká písmena: např. „<Příklad>“ a „</Příklad>“ je pár, který vyhovuje správně strukturovanému dokumentu, pár „<Příklad>“ a „</příklad>“ je chybný.

Jednoduchý recept v XML jako příklad by mohl vypadat takto [18]:

```
<?xml version="1.0" encoding="UTF-8"?>
<recept jméno="chleba" čas_přípravy="5 minut" čas_vaření="3
hodiny">
      <titulek>Jednoduchý chleba</titulek>
       <přísada množství="3" jednotka="šálky">Mouka</přísada>
       <přísada množství="0,25" jednotka="unce">Kvasnice</přísada>
       <přísada množství="1,5" jednotka="šálku">Horká voda</přísada>
       <přísada množství="1" jednotka="kávová lžička">Sůl</přísada>
       <instrukce>
             <krok>Smíchejte všechny přísady dohromady a dobře
             prohněťte.</krok>
             <krok>Zakryjte tkaninou a nechejte hodinu v teplé
             místnosti.</krok>
             <krok>Znovu prohněťte, umístěte na plech a pečte v
             troubě.</krok>
       </instrukce>
</recept>
```

5.4 Kontrola struktury dokumentu XML

XML neobsahuje předdefinované značky (tagy), je třeba definovat vlastní značky, které budeme používat. Tyto značky je možné (nepovinně) definovat v souboru DTD. Potom je možné automaticky kontrolovat, zda vytvářený XML dokument odpovídá této definici. Program, který tyto kontroly provádí, se nazývá parser. Při vývoji aplikací můžeme parser použít, a ten za nás detekuje většinu chyb v datech [18].

DTD není jediný definiční jazyk pro XML. Neobsahuje možnost kontrolovat typy dat (čísla, měnové údaje, údaje o datu a čase). To je vlastnost, která chybí při zpracování dat databázového charakteru. V současné době se pod názvem XML schémata pracuje na půdě konsorcia W3C na vytvoření jednotného standardu, který tyto kontroly umožní [18].

Pro různé standardní aplikace byla postupně vytvořena schémata, která definují značky (názvy elementů) pro konkrétní typy dokumentů. Příkladem může být DocBook, který definuje struktury pro vytváření knih, článků, vědeckých publikací a podobně. Výhodou takových aplikací je, že současně s definičními soubory DTD je dodávána sada stylů (XSL souborů) pro následné zpracování a přípravu pro tisk, nebo pro převod do jiných standardních tvarů (PostScript, HTML atd.) [18].

Další vlastností XML je, že v jednom dokumentu můžeme používat najednou nezávisle na sobě několik druhů značkovaní pomocí jmenných prostorů (namespaces). To umožňuje kombinovat v jednom dokumentů několik různých definic ve formě DTD nebo schémat bez konfliktů v pojmenování elementů [18].

Definice DTD souboru se pak provádí hned po deklarací XML:

```
<?xml version="1.0"?>
<!DOCTYPE dopis SYSTEM "dopis.dtd">
<dopis>
      <adresát>
      <jméno>Jan Novák</jméno>
      <adresa>...</adresa>
      </adresát>
      <předmět>Pozvánka na besedu o XML</předmět>
      <tělo>
            <para>Vážený pane,</para>
            <para>rád bych Vás jménem naší...</para>
      </tělo>
</dopis>
```

Druhá řádka ukázkového XML-dokumentu říká, že definice DTD je uložena v souboru dopis.dtd. DTD definuje elementy a atributy, které mohou být v dokumentu daného typu použity. V DTD pro psaní dopisů by nadefinovalo, že dopis se skládá

z informaci o adresátovi, z předmětu dopisu a z jeho těla. U adresáta se může zadat jméno a adresa. Tělo dopisu se skládá z jednotlivých odstavců textu [7].

V DTD se definuje jaké jsou přípustné vzájemné kombinace elementů, které elementy mohou být vynechány, které se mohou opakovat atd. Pokud je XML dokument spojen s nějakým DTD, může se snadno pomocí parseru zjistit, zda dokument splňuje požadavky definované v DTD [7].

II. PRAKTICKÁ ČÁST

6 PROGRAM MILIONÁŘ

Smyslem praktické části této diplomové práce byla názorná ukázka práce s texty v programech pro jazykovou lokalizaci. K tomu účelu byla naprogramována počítačová hra Milionář inspirovaná televizní soutěží Chcete být milionářem? [11a].

6.1 Výběr téma programu

Televizní soutěž Chcete být milionářem? [11a] je známá po celém světě a je velice oblíbená již několik let. Je to vědomostní soutěž, ve které soutěžící odpovídá na otázky týkající se nejrůznějších témat. Nikdo však nemůže znát odpověď na vše a proto má soutěžící na výběr ze čtyř možných odpovědí, ale jen jedna z nich je správná. Dále má na pomoc několik nápověd, které se liší podle verze soutěže, jelikož za těch pár let už prošla několika změnami. Soutěž tedy obsahuje mnoho textů což bylo hlavním důvodem proč byla zvolena jako téma ke zpracování do její programové podoby.

Tato hra již existuje ve spoustě programových podob jako jsou online hry, java hry pro mobilní telefony, stolní hry a běžné aplikace, ale vždy se dá vymyslet něco nového.

6.2 Odlišnost od ostatních verzí

Jak už bylo řečeno hra již má mnoho programových verzí a všechny jsou si téměř podobné. Aby hra Milionář přinesla i něco nového odlišuje se od jejich předchůdců několika způsoby.

Všechny předchozí verze mají jednu závažnou chybu, která se schovává v nápovědě Diváci. I když se použije na jakkoliv složitou otázku, vždy poradí správně a s velkou pravděpodobností. Tento nedostatek byl tedy odstraněn a úprava se týkala i ostatních nápověd.

Jelikož byla hra vytvořena podle nové podoby české televizní soutěže přibyla i nová nápověda Výměna otázky.

Největší odlišností je pak to, že lidé si mohou sami do této hry přidávat otázky, jelikož jsou uloženy v externím snadno editovatelném souboru. Postupem času tak může tato hra

obsahovat velké množství otázek. Soubor s otázkami pak může být třeba volně ke stažení na internetu a ostatní lidé si tak budou moci svůj program obohatit o nové otázky.

Program může obsahovat několik jazykových verzí uložených v samostatných externích souborech. Program i externí soubory jsou tedy proto napsány pomocí znakové sady Unicode, která umožňuje správné zobrazení znaků specifických pro daný jazyk.

Program je napsán pomocí multiplaformní knihovny a je tedy možné ho překompilovat tak aby běžel i na jiných softwarových platformách něž je Windows.

7 TVORBA PROGRAMU

Program byl napsán v programovacím jazyce C++ s pomocí multiplatformní knihovny wxWidgets verze 2.8.7 [10]. K tomu bylo použito vývojové prostředí Code::Blocks verze 8.02 [15]. Mezi lidmi jsou při trávení volné chvíle čím dál více oblíbené okenní aplikace typu Soliter, Spider Soliter a Srdce proto i program Milionář byl navrhnut jako jednoduchá okenní aplikace. Program i jeho externí textové soubory typu XML byly napsány znakovou sadou Unicode.

7.1 Struktura souborů XML

Pro uchovávání textů programu byl zvolen formát souborů XML kvůli jeho vlastnostem a to především kvůli tomu, že tagy nejsou předem předefinovány a tak mohou byt nazvány dle potřeby, dále pak díky jeho velké podpoře v knihovně wxWidgets a možnosti použití znakové sady Unicode.

Tyto soubory neobsahují jen otázky a odpovědi, ale také popisky menu, návod hry, druh světového jazyka a jiné texty použité v programu. Struktura tagů v XML souboru byla zvolena podle rozdělení složitosti otázek v programu.

Otázky jsou rozděleny do kategorii podle jejich hodnoty a uzavřeny mezi tagy které označují příslušnou kategorii. Tagy kategorií jsou označeny názvem a číslem nejvyšší hodnoty otázky v kategorii „<otázky nejvyšší hodnota>“. Otázky jsou pak uzavřeny v tagu označující jejich pořadí v kategorii „<o číslo pořadí>“. Toto číslo však má pouze informativní charakter a není nutné jej uvádět. Tyto tagy se tedy mohou pojmenovat jak je uznáno za vhodné a mohou se i opakovat, na funkčnost programu to nemá žádný vliv. Otázky jsou z každé kategorie vybírány náhodně a tak jejich pořadí v kategorii nemá také žádný význam. Mezi tagy označující pořadí otázky už jsou uzavřeny konkrétní otázky s k nimi přiřazenými možnosti odpovědi. Aby program poznal co je otázka a co odpovědi, a která z nich je správná jsou opět rozděleny do tagů dle významu. Tři špatné odpovědi jsou od sebe rozděleny posledním písmenem v názvu tagu „a,b,c“. Pořadí špatných odpovědí nemá žádný význam jelikož program pořadí odpovědí generuje náhodně. Zde se však ani jeden z názvů tagů nesmí zaměnit, program by jinak pojmenovaný tag nepřečetl a

otázka by tak byla neúplná. Takto například vypadá otázka, která se zobrazuje v hodnotách od 1000 do 3000 a je v kategorii uvedena jako první:

Kategorie *<otazky3>*

Pořadí otázky *<o1>*

<otazka>Jak se jmenoval kamarád Ferdy Mravence?</otazka>

<spravne>Brouk Pytlík</spravne>

<blbea>Hurvínek</blbea>

<blbeb>Brouk Střevlík</blbeb>

<blbec>Beruška</blbec>

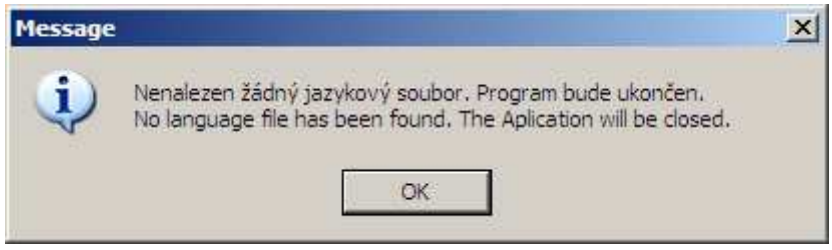
</o1>

</otazky3>

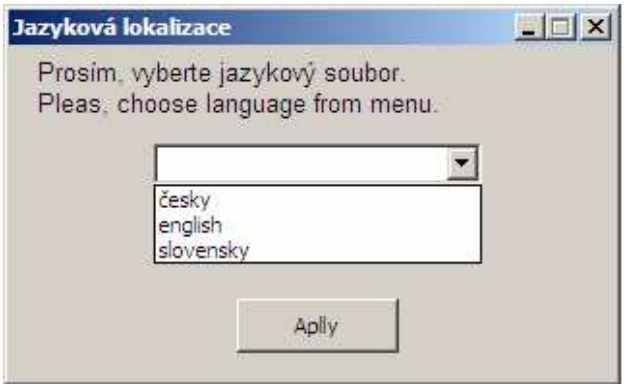
Viz. přiložené XML soubory na cd v adresáři složka_hry\langs\.

7.2 Načítání XML souborů

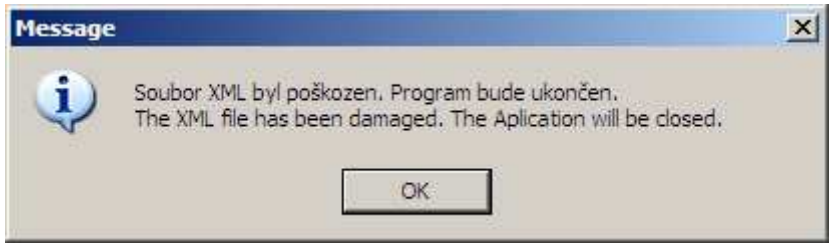
Program hned po spuštění hledá v pevně nastaveném adresáři složka_hry\langs\ soubory XML. Pokud není žádný nalezen zobrazí se okno se zprávou (Obr. 1) a program je ukončen. Pokud je nalezen jen jeden soubor načte se automaticky a spustí se hra. Když je XML souborů více zobrazí se nejprve dialogové okno pro výběr jazykového souboru (Obr. 2). Po vybrání jazykového souboru se tento soubor načte a spustí se hra. Při načítání XML souboru se zároveň kontroluje zda soubor obsahuje všechny důležité informace. Pokud při načítání souboru nastane chyba, je zobrazeno okno se zprávou o chybě (Obr. 3) a program je ukončen. Jelikož tyto chyby vznikají dříve než je možné načíst jazykový soubor, jsou popisky chyb uloženy přímo ve zdrojovém kódu programu a nemohou být měněny. Z tohoto důvodu jsou alespoň popisky dvoujazyčné.



Obr. 1. Upozornění na nenalezení XML souboru



Obr. 2. Dialog pro výběr jazykového souboru



Obr. 3. Upozornění na chybu při načítání souboru

7.2.1 Vypsání jazykových verzí XML souborů

Při zjištění většího počtu XML souboru je přerušeno vykreslování hlavního okna hry (Obr. 6) a je zavolána funkce, která zobrazí dialogové okno pro výběr jazykové lokalizace (jazykového souboru xml) a čeká dokud v tomto dialogovém okně není zmáčknuto tlačítko Aplly (Obr. 2). Program poté zjistí který soubor byl vybrán a předá jej dalším funkcím ke zpracování. Pokud není žádný soubor vybrán program jej chápe jako chybný a zobrazí zprávu o chybě (Obr. 3) a ukončí program.

Dialogové okno pro výběr je tvořeno pouze prvkem Choice, do kterého se vypisují jazykové lokalizace souborů a tlačítkem Aplly, kterým je potvrzen výběr. Druhy jazyků jsou do prvku Choice přiřazovány postupně ve smyčce. Nejprve se načte první soubor a zněj se přečte textová hodnota uzavřena mezi tagy <lang>jazyk souboru</lang>. A takto se to opakuje dokud nejsou přečteny všechny soubory. Hlavní okno programu má tyto soubory uloženy v dynamickém poli takže stačí když mu dialogové okno předá index (číslo) určující pozici prvku v poli, na kterém je uložena cesta k souboru.

V případě nalezení jen jednoho souboru je postup stejný akorát se vynechá volání dialogového okna pro výběr jazyka a funkcím pro zpracování souboru je předán přímo index 0, na kterém je cesta k souboru v poli uložena.

7.2.2 Načtení popisků menu a herních hlášení

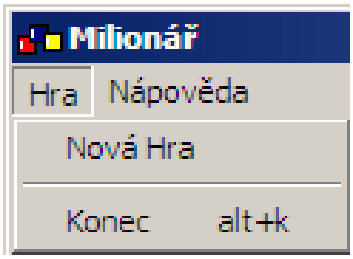
Vybraný jazykový soubor je předán funkci pro načtení popisků menu. Načtení popisků se musí provádět před vykreslením hlavního okna programu jelikož při vytváření menu již musí být k dispozici. Bez popisků menu a herních textů by nebyla hra plně ovladatelná a proto se zároveň provádí kontrola zda byly načteny všechny popisky a texty hry. V případě, že vše proběhne v pořádku je zavolána funkce pro načtení otázek.

Popisky menu jsou uloženy v globálním dynamickém poli as_gui a herní hlášení v globálním dynamickém poli as_texty. Aby se zabránilo chybám při případném přeházení posloupnosti tagů v XML souboru je nalezený popisek/text vždy přiřazen na jemu pevně danou pozici v poli. Z tohoto důvodu je polím nejprve alokována paměť podle počtu popisků/textů. Kontrola správnosti načtení všech popisků a textů je prováděna jednoduchou inkriminací proměnné typu integer, která by se na konci měla rovnat 13. Toto číslo se totiž rovná součtu všech popisků menu a herních textů.

Jako příklad je uveden zkrácený výpis, který ukazuje načtení textu „Určitě je to možnost“ pro nápovědu Přítel na telefonu. Nejprve je alokována paměť pro pole herních textů, poté se nalezne tag označující nápovědu Přítel na telefonu a v něm pak načtení hodnoty tagu označující odpověď „Určitě je to možnost“:

```
as_texty.Alloc(5);                                  alokace paměti
if(potomek->GetName()== wxT("telefon")){            zjištění tagu nápovědy po telefonu
    pot = potomek->GetChildren();                   předání potomka tohoto tagu
    prvek++;                                        přičtení nalezení textu
    if(pot->GetName()==wxT("urcite"))               pokud je název potomka „urcite“ načte
    as_texty.Insert(pot->GetNodeContent(),0);}      se text do pole na pozici 0
```

Externí popisky menu ne jen, že umožňují jejich libovolné přejmenování nebo přeložení do jiného jazyka, ale také umožňuje změnu jejich ovládání pomocí klávesových zkratek. Například v českém jazyce je klávesová zkratka pro konec „alt+k“ (Obr. 4), anglickém „alt+x“ jako exit (Obr. 5) a v jiných jazycích může být opět jiná.



Obr. 4. Menu CZ

Obr. 5. Menu AN

7.2.3 Načtení otázek

Načítání otázek je velice podobné načítání popisků menu a herním textů. Otázky jsou načítány do dynamického pole objektů. Toto pole se chová podle stejných principů a pravidel jako standardní dynamické pole s tím rozdílem, že se k prvkům, které uchovává, chová jako k objektům a ne jako k hodnotám. Toto pole má také svou třídu deklarovanou v hlavičkovém souboru MilionarMain.h. Třída pole obsahuje několik proměnných typu *wxString,* do kterých jsou načítaný otázky a odpovědi. K ukládání otázek a odpovědí tedy stačí jedno pole přičemž otázka a její odpovědi mají stejnou hodnotu prvku pole (index) jen jsou uchovány v jiné proměnné pole. Při ukládání otázky pak příkaz vypadá asi takto:

p_zaznam – dynamické pole objektů

a_ot10m – globální dynamické pole objektů, ve kterém jsou uloženy všechny otázky za 5 a 10 miliónů

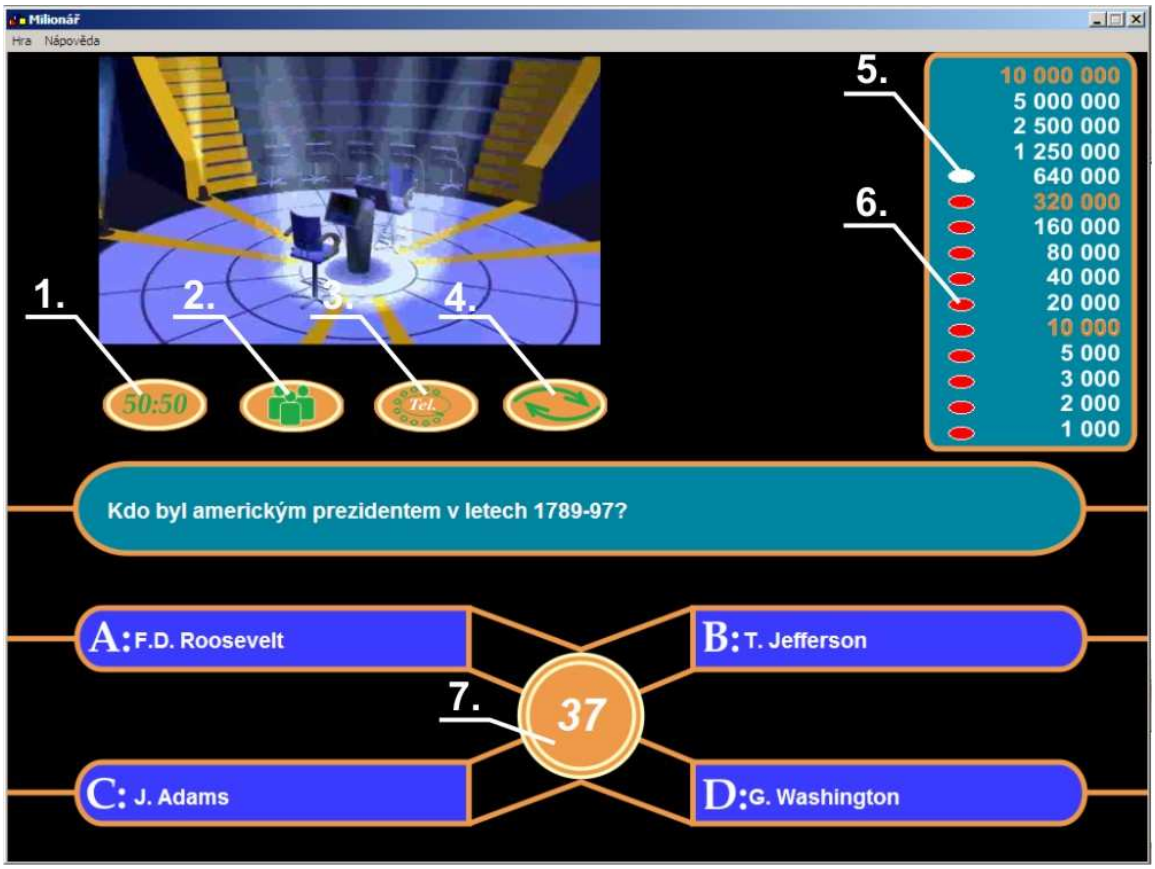
ot_od – uzel (tag) XML souboru

```
if(ot_od->GetName()== wxT("otazka")) p_zaznam.otazka = ot_od->GetNodeContent();
if(ot_od->GetName()== wxT("spravne"))p_zaznam.od_true = ot_od->GetNodeContent();
if(ot_od->GetName()== wxT("blbea")) p_zaznam.od_blba1 = ot_od->GetNodeContent();
if(ot_od->GetName()== wxT("blbeb")) p_zaznam.od_blba2 = ot_od->GetNodeContent();
if(ot_od->GetName()== wxT("blbec")) p_zaznam.od_blba3 = ot_od->GetNodeContent();
a_ot10m.Add(p_zaznam);
```

Po načtení všech otázek je zjišťováno zda každá z kategorii má alespoň čtyři otázky. Zjistí se to pomocí funkce *Count()*, která vrátí počet prvků pole. Kategorie musí mít minimálně čtyři otázky jelikož je to maximum, které se během jedné hry může v jedné kategorii zobrazit. Běžně se zobrazují tři, ale při použití nápovědy „Výměna otázky“ se zobrazovaná otázka zahodí a musí se načíst nová otázka, patřící do kategorie o stejné obtížnosti. Pokud jedno z polí (kategorie) neobsahuje alespoň tyto čtyři otázky, je zobrazeno okno se zprávou o špatném souboru (Obr. 3) a hra je ukončena. Pokud je však vše v pořádku, už nic nebrání vykreslení hlavního okna programu a tím spuštění hry.

7.3 Hlavní okno programu

Hlavni okno programu (Obr. 6) se stará o celý běh hry. Na jeho pozadí je vykreslena bitmapa znázorňující ohraničení jednotlivých zobrazovacích a ovládacích prvků hry. Pro výpis otázek, odpovědí a zbývajícího času jsou použity prvky wxStaticText nastaveny na readonly. Tlačítka nápovědy jsou vytvořeny pomocí prvku wxBitmapButon. Elipsy zobrazující správně zodpovězené otázky a aktuální otázku jsou vykreslovány přímo programem.



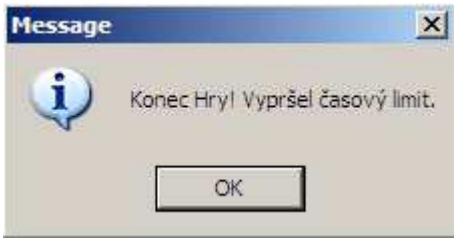
Obr. 6. Hlavní okno programu s popisem

Legenda k obrázku:

1. Tlačítko nápovědy 50:50
2. Nápověda Diváci
3. Nápověda Telefon

4. Nápověda Výměna otázky
5. Označení aktuální otázky
6. Označení správně zodpovězené otázky
7. Čas zbývající na zodpovězení otázky

K odpočtu zbývajícího času je použit časovač nastavený na 1 sekundu, který během jednoho cyklu provádí dekrementaci předem nastavené proměnné a její hodnotu pak vypíše do prvku wxStaticText určenému pro vypisování času. V případě, že již čas vypršel ukončí se hra a vykreslí se okno se zprávou informující hráče o uplynutí času (Obr. 7). Pak se provede kontrola zda má hráč nárok na jednu z odměn. Pokud ano, vykreslí se šek s hodnotou výhry (Obr. 8).



Obr. 7. Vypršení časového limitu

Obr. 8. Šek s vyhranou částkou

7.4 Nápověda 50 na 50

Nápověda 50:50 (Obr. 6) je jednoduchou nápovědou kdy zmizí dvě špatné odpovědi a zůstane jen správná odpověď a jedna odpověď špatná. Špatná odpověď, která zůstane zobrazena se generuje náhodně a tak tedy při každém zobrazení stejné otázky může zůstat jiná špatná odpověď. Špatné odpovědi, které zmizí už nesmějí být aktivní pokud se na ně klikne myší proto je se u nich nastaví parametr *FALSE*.

7.5 Nápověda Diváci

Nápověda Diváci (Obr. 6 a Obr. 9) přiřadí každé odpovědi pravděpodobnost v procentech, že daná odpověď je správná. Tato nápověda má simulovat publikum v herním sále, kdy každý divák zvolí jednu z odpovědí, o které si myslí, že je správná. Se složitostí otázky samozřejmě i lidé v publiku přestávají vědět správnou odpověď a už jen tipují a tomu je nápověda přizpůsobena.

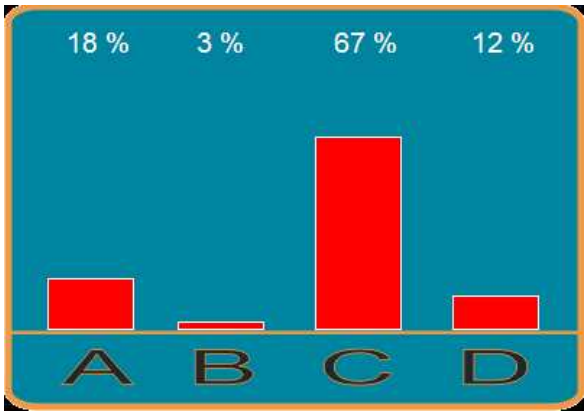
Při použití nápovědy se nejprve zjistí jakou má otázka obtížnost podle jejího pořadí a podle toho je vygenerovaná pravděpodobnost správné odpovědi. Pokud je to otázka maximálně pátá v pořadí tedy v hodnotě maximálně 10 000, je nastaven rozsah generátoru mezi 70 až 90%. Tím je zajištěno, že správná odpověď bude mít vždy největší pravděpodobnost.

Pokud je otázka šestá až desátá v pořadí (max. 320 000) je nastaven rozsah mezi 30 až 70%. Pravděpodobnost, že správná odpověď bude mít nejvíce procent je zde už jen 50:50.

Pokud už je otázka více jak desátá v pořadí je nastaven rozsah od 5 do 60%. Pravděpodobnost, že správná odpověď bude mít nejvíce procent je mizivá.

Pravděpodobnost ostatních odpovědí je generována tak, že se od 100 odečte vygenerovaná pravděpodobnost správné odpovědi a výsledná hodnota se vloží jako maximální hodnota pro generování. Vygenerovaná hodnota je pak přiřazena jedné z odpovědí a celé se to opakuje pro další odpověď s tím, že před odečtením se sečtou vygenerované hodnoty pravděpodobnosti otázek.

Vygenerované hodnoty pravděpodobnosti správnosti otázek se poté vykreslí do sloupcového grafu i s přesně uvedenými hodnotami v procentech (Obr. 9).



Obr. 9. Nápověda Diváci při 7 otázce v hodnotě 40 000

7.6 Nápověda Telefon

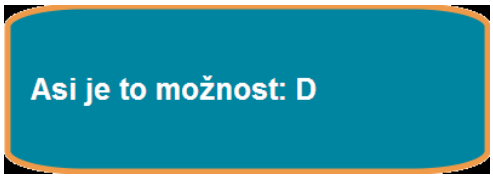
Tato nápověda (Obr. 6 a Obr. 10) simuluje přítele na telefonu. V externím souboru XML jsou uloženy tři možné odpovědi, které se načtou a zobrazí se podle vygenerování pravděpodobnosti podobně jako u nápovědy Diváci.

Pokud je pořadí otázky do páté pozice automaticky se vypíše správná odpověď ve znění: „Ahoj, určitě je to možnost: X“.

Pokud je pořadí otázky od šesté do desáté pozice je vygenerovaní správné odpovědi v rozmezí 50 až 100%. Pokud je vygenerovaná pravděpodobnost větší něž 70% je opět vypsána správná odpověď jako v prvním případě. Pokud je však vygenerovaná hodnota pod 70% náhodně se vybere mezi správnou a jednou špatnou odpovědí v poměru 50:50. Odpověď je tedy ve tvaru: „Asi je to možnost: X“, ale za X může být dosazena jak správná tak špatná odpověď.

Poslední možností je otázka, která je větší než desátá v pořadí. Zde už se generuje pravděpodobnost jen do 60%. První varianta správné odpovědi je tady nemožná. Pokud je vygenerovaná hodnota pravděpodobnosti větší než 30% opět voleno mezi správnou a

špatnou odpovědí jako tomu bylo ve druhém případě. Pokud je vygenerovaná hodnota pod 30% vypíše se odpověď: „Ahoj, promiň tohle nevím.“.



Obr. 10. Ukázka nápovědy telefon

7.7 Nápověda Výměna otázky

Nápověda Výměna otázky (Obr. 6) je k dispozici až po překročení hranice 320 000 neboli desáté otázky. Do této doby není tlačítko k nápovědě vidět. Nápověda nedělá nic jiného něž že zahodí aktuální otázku a vygeneruje náhodně novou otázku stejné obtížnosti a nastaví čas opět na 40 vteřin.

7.8 Označení odpovědi

Zachytávání vybrání odpovědi respektive kliknutí na odpověď muselo být z důvodu použití prvku wxStaticText vytvořeno několika způsoby. Jeden ze způsobů je po každém kliknutí levým tlačítkem myši zjišťovat pozici kurzoru a druhý způsob je kliknutím přímo na text odpovědi zjistit odpověď.

7.8.1 Zjišťování pozice kurzoru myši

Pokud se myší klikne kdekoliv v okně programu kromě vložených prvků jako jsou třeba tlačítka, menu a podobně vyvolá se událost EVT_LEFT_DOWN. V tabulce událostí je pak této události přiřazena funkce *onmouse()*, která ji zpracuje. Tato funkce zjistí pozici X a Y kurzoru myši při stlačení levého tlačítka. Poté porovnává zda je souřadnice X v rozmezí některé ze čtyř odpovědí. Pokud ano porovná zda i souřadnice Y odpovídá rozsahu jedné z odpovědí, které odpovídala souřadnice X.

Pokud tedy souřadnice X i Y odpovídají rozsahu souřadnic jedné z odpovědí nastaví se tato odpověď jako vybraná. Poté se provede kontrola zda je vybraná odpověď aktivní. Tedy jestli nabyla použita nápověda 50:50 a odpověď nebyla vymazána a nebyl ji nastaven parametr *FALSE*. Pokud byl nastaven parametr *FALSE* hra pokračuje dál jako by se nic nestalo. V opačném případě se porovná vybraná odpověď se správnou odpovědí a provede se příslušná akce podle toho zda se odpovědi rovnají či nikoliv.

Pokud souřadnice X a Y neodpovídají rozsahu ani jedné z odpovědí hra pokračuje dál jako by se nic nestalo.

Tento z působ tedy funguje v případě, že se klikne do prostoru otázky vymezeného pomocí bitmapy na pozadí tam kde nezasahuje text odpovědi a tedy možné kliknou přímo na prvek wxStaticText.

7.8.2 Kliknutí na prvek wxStaticText

Kliknutím na nějaký prvek se také vyvolá událost EVT_LEFT_DOWN, ale takováto událost je předána ke zpracování prvku, na který bylo kliknuto. V případě tlačítek jsou tyto funkce taky mapovány pomocí tabulky událostí. V případě použití prvku wxStaticText to však možné není. Tím tedy není spuštěna funkce na zachytávání polohy myši a porovnání její pozice s rozsahem souřadnic odpovědí *onmouse()*.

Byla proto použita funkce *Connect()*, která umožňuje událost vyvolanou nějakým prvkem předat ukazatel ke zpracování funkci jiné. Ovšem předání obsluhy této události nemohlo být předáno ani funkci *onmouse()*, protože souřadnice pozice kurzoru myši by se nevztahovaly k rozměrům okna, ale k prvku který událost vyvolal. Namísto toho aby tedy byly vráceny například souřadnice X=120, Y=500 vztahující se k oknu, byly by vráceny souřadnice X=10, Y=5 vztahující se k rozměrům a umístění prvku. Funkce by tedy proběhla jako by se kliklo mimo nějakou odpověď. Proto byly vytvořeny čtyři stejné funkce, každá pro jednu odpověď, která se postará o označení odpovědi a kontroly zda je odpověď aktivní a zda je správná. Poté se provedou stejné akce jako při zjišťování pozice kurzoru myši.

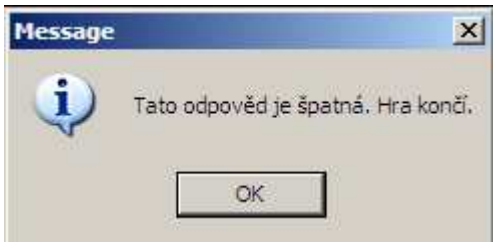
Příklad ukazuje použití funkce *Connect()* na předání ukazatele na obsluhu události po kliknutí na odpověď A funkci *onClickOdA*:

```
wxStaticText *OdA;

void onClickOdA(wxMouseEvent& event);

OdA->Connect(wxEVT_LEFT_DOWN,wxMouseEventHandler(MilionarFrame::onClickOdA),
NULL, this);
```

7.8.3 Porovnání odpovědí

Po vybrání otázky se provádí kontrola zda vybraná odpověď je správná. V případě, že vybraná odpověď je správná vybere se další otázka nebo pokud byla otázka poslední, tedy za 15 miliónů, provede se vykreslení šeku a ukončení hry.

V případě, že vybraná odpověď byla špatná, provede se stejná akce jako při vypršení časového limitu. Vykreslí se okno s upozorněním na špatnou odpověď (Obr. 11), zkontroluje se zda má hráč právo na nějakou výhru a pokud ano zobrazí se šek.



Obr. 11. Informace o špatné odpovědi

8 PŘÍKLAD NAČTENÍ XML SOUBORU

Zde je popsán zdrojový kód načtení XML souboru bez rozdělení do více funkcí. Pro jednoduchost je kód bez jakékoliv kontroly správnosti. Kód ukazuje načtení popisků menu záložky HRA.

```
as_gui.Alloc(9);       //alokace dynamického pole wxArrayString
wxXmlDocument x_file;  //proměnná uchovávající xml dokument
wxXmlNode *potomek;    //ukazatel na uzel (tag)
wxXmlNode *pot;  //ukazatel na uzel (tag), bude pouzit na potomka

//nacteni xml souboru s kódováním UNICODE
x_file.Load(wxT("langs\cz.xml",wxT("UTF-8"));
x_file.GetRoot()->GetName();     //zjištění hlavního tagu (uzlu) <root>

//první potom uzlu <root> vráti <lang>
potomek=x_file.GetRoot()->GetChildren();

//prochazeni tagu a ukladani hodnot (popisku) do pameti
while (potomek){
   //nacteni popisku menu "HRA"
 if (potomek->GetName()== wxT("hra")){
   pot = potomek->GetChildren(); //zjistění potomka tagu <hra>
     while (pot){
        //nacte text mezi taga a ulozi na příslušnou pozici v poli
    if (pot->GetName()== wxT("title"))
        as_gui.Insert(pot->GetNodeContent(),1);
        if (pot->GetName()== wxT("nova"))
        as_gui.Insert(pot->GetNodeContent(),2);
        if (pot->GetName()== wxT("jmeno"))
        as_gui.Insert(pot->GetNodeContent(),3);
        if (pot->GetName()== wxT("score"))
        as_gui.Insert(pot->GetNodeContent(),4);
        if (pot->GetName()== wxT("exit"))
        as_gui.Insert(pot->GetNodeContent(),5);
         pot=pot->GetNext();
      }          //konec while(pot)
 }               //konec if(potomek->GetName()== wxT("hra"))
 potomek = potomek->GetNext();//najde se další potomek uzlu <root>
}                //konec while(potomek)
```

ZÁVĚR

V teoretické části této diplomové práce byla popsána znaková sada ASCII kde jí zároveň byly vytyčeny její nedostatky týkající se dnešní doby. Dále byla popsána znaková sada Unicode, která nejen že odstraňuje nedostatky znakové sady ASCII, ale odstraňuje problémy mezi všemi znakovými sadami. Zároveň byly popsány její výhody týkající se nejen jejího použití v oblasti webové tvorby. Aby byl její popis spravedlivý vůči ostatním znakovým sadám bylo ji vytknuto i několik nedostatků. Poté byla stručně popsána multiplatformní knihovna wxWidgets, se kterou se pracovalo v praktické části. Dále byly popsány způsoby, kterými se uchovávají texty programů ať už přímo ve zdrojovém souboru nebo v externím souboru mimo program, což umožňuje jejich přeložení do jiného jazyka. Ke každému způsobu byl zároveň uveden jeden program, který tento způsob ukládání textu využívá. Nakonec byl popsán značkovací jazyk XML, která v poslední době zaznamenal velký rozmach a také byl použit v praktické časti této práce.

V praktické části byl pak vytvořen program Milionář podle televizní soutěže „Chcete být milionářem?“. Byl navržen tak aby mohl být přeložen do jakéhokoliv světového jazyka, mohl být překompilován tak aby byl spustitelný na několika různých softwarových platformách a otázky si lidé mohli dopisovat samy bez nutnosti umět programovat a mít k tomu speciální program. Toho bylo dosáhnuto použitím znakové sady Unicode, která umožňuje správné zobrazení všech světových jazyků. Použitím multiplaformí knihovny wxWidgets při vytvářená programového kódu a jeho kompilaci. Dále pak použitím externího souboru psaného pomocí jazyka XML ve znakové sadě Unicode, který umožňuje uložení textů programu a otázek mimo zdrojové kódy programu. Tento soubor je navíc snadno editovatelný v jakémkoliv textovém editoru dokonce i v programu Notepad. Tím byla vytvořena názorná ukázka práce s texty pro snadnou jazykovou lokalizaci. V programu ovšem chybí zvuky, animace a ukládání score pro větší prožitek ze hry.

SEZNAM POUŽITÉ LITERATURY

[1] 2WAYTRAFFIC, Milionář [online]. 2008. Dostupný z WWW: <http://milionar.atlas.cz/soutez/>.

[2] BLIŽŇÁK, Michal. Systémové programování. Zlín : UTB ve Zlíně, 2005

[3] DOLEČEK, Jaromír, Standart Unicode [online]. 1998. Dostupný z WWW: <http://www.ics.muni.cz/bulletin/articles/133.html>.

[4] DT COMMUNITY, Daemon Tools [online]. 2008. Dostupný z WWW: <http://www.daemon-tools.cc/dtcc/announcements.php/>.

[5] HAŽ Jarda, Becherovka game [online]. 2007. Dostupný z WWW: <http://game.becherovka.cz/>.

[6] CHALUPA, Radek. Pár slov o UNICODE [online]. 2002. Dostupný z WWW: <http://www.builder.cz/art/cpp/winapi5.html>.

[7] KOSEK Jiří, XML [online]. 2000. Dostupný z WWW: <http://www.kosek.cz/clanky/xml/xml-uvod.html/>.

[8] MNEMONIC, Jak připravit hru na snadnou lokalizaci – 1. díl [online]. 2007. Dostupný z WWW: <http://www.ceske-hry.cz/forum/viewtopic.php?p=2887&sid=420cfb126205b4b902a7ad5a69d457e5>.

[9] MORRISON, Michael. Naučte se programovat počítačové hry za 24 hodin. 1. vyd. Brno : Computer Press, 2004. 421 s. ISBN 80-251-0371-4.

[10] Ollivier Kevin, wxWidgets [online]. 2007. Dostupný z WWW: <http://www.wxwidgets.org>.

[11] PEPR SOFT, Total Commander [online]. 2008. Dostupný z WWW: <http://www.totalcommander.cz/>.

[12] PETZOLD, Charles. Programování ve Windows. 1. vyd. Praha : Computer Press, 1999. 1216 s. ISBN 80-7226-206-8.

[13] REALNETWORKS, RealPlayer [online]. 2008. Dostupný z WWW: <http://www.realplayer.com/>.

[14] ROŽÁNEK, Filip, GERALT. Jak překládat? [online]. 2001. Dostupný z WWW: <http://cestiny.idnes.cz/prekladani/>.

[15] The Code::Blocks Team, Code::Blocks [online]. 2008. Dostupný z WWW: <http://www.codeblocks.org>.

[16] UNICODE, Inc. What is Unicode? [online]. 2007. Dostupný z WWW: <http://www.unicode.org/standard/WhatIsUnicode.html>.

[17] WIKIPEDIA, ASCII [online]. 2008. Dostupný z WWW: <http://cs.wikipedia.org/wiki/ASCII >.

[18] WIKIPEDIA, Extensible Markup Language [online]. 2008. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Extensible_Markup_Language/>.

[19] WOLNY, Jiří, Znaková sada Unicode a UTF-8 [online]. 2005. Dostupný z WWW: <http://www.mvcr.cz/rady/faq/unicode/index.html>.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MFC	Microsoft Foundation Classes
GUI	Graphical User Interface
ANSI C	American National Standard for Programming Languages - C
ACII	American Standard Code for Information Interchange
XML	eXtensible Markup Language
API	Application Programming Interface
DTD	Document Type Definition
HTML	HyperText Markup Language
XSL	eXtensible Stylesheet Language

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM PŘÍLOH

PI: CD disk s aplikací Milionář, zdrojovými soubory a diplomovou prací.