Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Návrh konzolového frameworku pro UI a grafiku v C++

Jozef Marek

Bakalářská práce

2018

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jozef Marek**
Osobní číslo:	**I14338**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Návrh konzolového frameworku pro UI a grafiku v C++**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování:

Cílem bakalářské práce je vytvořit framework v jazyce C++, který bude podporovat tvorbu UI v konzolových aplikacích. Framework by rovněž měl umožnit uživateli používat ?grafický? výstup přímo na konzoli.

V teoretické části práce budou popsány možnosti standardní konzole na Windows. Dále budou popsány funkcionality z Windows API, které je možné použít pro rozšířenou práci s konzolí. Bude proveden návrh knihovny a jejího API, tak aby použití knihovny bylo co nejjednodušší.

V praktické části bude vytvořen framework (statická knihovna), která bude umožňovat tvorbu základních UI dialogů v konzoli (input box, message box, jednoduché menu na výběr z několika položek) a také přímý výstup znaků a barev do konzole. Funkčnost a možnosti knihovny budou demonstrovány na několika jednoduchých příkladech.

Rozsah grafických prací:

Rozsah pracovní zprávy: **30-40 stran**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

PRATA, Stephen. Mistrovství v C++. 4., aktualiz. vyd. Brno: Computer Press, 2013, 1176 s. Bestseller (Computer Press). ISBN 978-80-251-3828-1.

PETZOLD, Charles. Programming Windows. 5th ed. Redmond, Wash.: Microsoft Press, c1999. ISBN 157231995X.

Vedoucí bakalářské práce: **Ing. Roman Diviš**
Katedra softwarových technologií

Datum zadání bakalářské práce: **31. října 2016**

Termín odevzdání bakalářské práce: **12. května 2017**

Univerzita Pardubice Česká republika

L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

Mgr. Josef Horálek, Ph.D.
vedoucí katedry

V Pardubicích dne 31. března 2017

Prohlášení autora

Prohlašuji, že jsem tuto práci vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 10. 12. 2018

podpis autora

Jozef Marek

PODĚKOVÁNÍ

Rád bych poděkoval Ing. Romanu Divišovi za cenné rady, věcné připomínky, trpělivost a vstřícnost při konzultacích a vypracování této práce. Dále děkuji mé rodině za trpělivost a oporu, které se mi dostávalo při tvorbě této práce. Nakonec bych chtěl poděkovat svým přátelům, a to zejména panu Ing. Liboru Šafkovi a Mgr. Veronice Glaserové za neuvěřitelnou podporu, která byla při psaní této práce zapotřebí.

ANOTACE

Práce se zabývá návrhem a implementací frameworku pro realizaci uživatelského rozhraní v textovém režimu (konzoli). V první části jsou představeny vybrané knihovny a frameworky s využitím pod různými operačními systémy. Dále je představeno Windows API a jeho využití v rámci textového (konzolového) režimu. Práce se dále zabývá návrhem a implementací vlastního objektového frameworku pro realizaci TUI.

KLÍČOVÁ SLOVA

C++, TUI, CLI, GUI, Windows API

TITLE

The console framework design for UI and C++ graphics

ANNOTATION

The bachelor thesis deals with the design and implementation of framework for the realisation of user interface in the text (console) mode. The first part introduces the selected libraries and frameworks used in various operating systems. Then Windows API and its use within the text (console) mode is presented. The thesis is further concerned with the design and implementation of self-made object framework for the realisation of TUI.

KEYWORDS

C++, TUI, CLI, GUI, Windows API

Obsah

SEZNAM ILUSTRACÍ

SEZNAM TABULEK

SEZNAM ZDROJOVÝCH KÓDŮ

SEZNAM ZKRATEK A ZNAČEK

TUI	Text User Interface
GUI	Graphics User Interface
CLI	Command Line Interface
API	Application Programming Interface
DLL	Dynamic Link Library
STL	Standard Template Library

1 Úvod

Cílem této práce je vytvořit framework, který zjednoduší tvorbu aplikací pod příkazovou řádkou s využitím UI, tzv. uživatelského rozhraní. Standardní konzole systémů nenabízejí bez jejich rozšíření, jako je to u Windows API, žádné grafické vstupy a výstupy. Proto systémy nabízejí možnost rozšíření, jako je to u operačního systému Windows a jeho API, vyvíjených aplikací nejen pro příkazovou řádku, ale i klasických desktopových aplikací. Tato rozšíření obsahují ovládací prvky nebo metody. Možnosti jejich využití je hned několik, a to buď přímého využití nebo použití některé z dostupných knihoven nebo frameworku třetích stran.

Začátky rozhraní v informatice dříve představovaly terminál s blikajícím kurzorem, který vyčkával na zadání příkazu a následné vykonání akce. Nebyla zde možnost barevných výstupů a formátování a velkým nedostatkem byla nutná znalost příkazů a jejich rozšířené funkce pomocí tzv. přepínačů. Postupem času se terminály a osobní počítače zdokonalovaly nejen ve výkonu hardwaru, ale právě i po softwarové stránce a tím i jejich rozhraní. Začaly se tvořit aplikace, které měly barevný výstup a formátování. Jejich ovládání se zjednodušilo na klávesové šipky zajišťující pohyb v aplikaci a výběr akce, která se nemusela psát ručně a nebyla nutná znalost k psaní příkazů. Tyto aplikace definovaly nové rozhraní TUI.

Dalším technickým krokem bylo představení GUI, které udělalo revoluci v ovládání počítačů. Toto uživatelské rozhraní a jeho používání přetrvává dodnes. Jedná se totiž o nejjednodušší ovládání počítačů, ke kterému postačí pouze myš a není nutná znalost příkazů. Nevýhodou tohoto rozhraní je velký nárok na výkon hardwaru.

2 User Interface

Uživatelské rozhraní neboli *User Interface* (UI) je způsob, jakým uživatel může ovládat chování systému, provádět jednoduché, ale i složité operace a následně získat jejich výsledky. Vstupní informace, které uživatel zadává mohou mít různou podobu. Od toho se definují tři základní skupiny rozhraní:

- CLI (*Command Line Interface*),
- TUI (*Text User Interface*),
- GUI (*Graphics User Interface*).

V aktuální době se nejčastěji setkáváme s třetí variantou tzv. GUI, která se rozšířila hlavně kvůli své jednoduchosti v ovládání systému a programového příslušenství. V ojedinělých případech se můžeme setkat i s předešlou verzí rozhraní (CLI), která se využívá spíše v zařízeních s málo výkonným hardwarem. Nevýhodou CLI je znalost příkazů na úrovni zkušeného uživatele či programátora. Názvy a použití příkazů se ovšem v závislosti na operačním systému liší. [1]

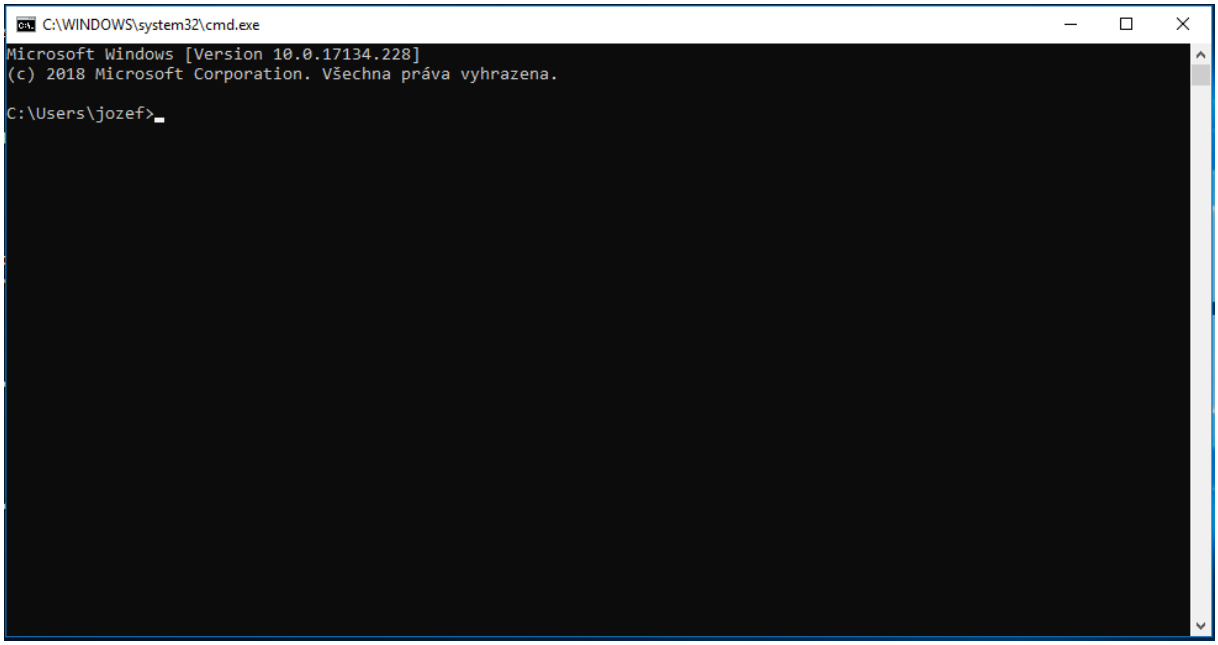
Příkazová řádka neboli CLI nabízí pod různými operačními systémy také rozšířené funkce, kterými jsou například podpora zobrazení barev v konzoli anebo vykreslování z bufferu. Tyto funkce se pak dají využít jako „grafický režim“ v konzoli. Toto rozšíření se často pojmenovává TUI (*Text User Interface*). Textové uživatelské rozhraní využívá speciálních znaků a barevného zvýraznění, které jako celek vytváří dojem grafického rozhraní. Dalšími možnostmi jsou ovládání programů v CLI pomocí kurzorových kláves nebo i ukazatele.

2.1 Druhy UI

2.1.1 CLI (Command Line Interface)

Příkazový řádek, který byl základem každého výpočetního stroje od terminálů až po osobní počítače, byl a je standardní výbavou. Ovládání tohoto řádku spočívá v zadávání příkazů a jejich následným potvrzením stiskem klávesy „Enter“. Jeho funkce postrádaly formátování vstupů a výstupů, barevné zvýrazňování nebo použití ukazatele neboli myši. Výhody tohoto řádku jsou především rychlejší zpřístupnění většiny služeb operačního systému a kombinace více příkazů najednou. Příkazový řádek je možné použít i v aktuálních verzích operačních systémů, jako jsou linuxové a unixové systémy nebo systém Windows, které již obsahují GUI.

Příkazový řádek zobrazuje výzvu k zadání příkazu, který může být popřípadě rozšířen o přepínače měnící funkci příkazu a vstupní parametry, pokud jsou zapotřebí. V linuxových a unixových systémech se tento řádek nejčastěji nazývá *shell*. V jiných systémech, jako je například DOS (*Disk Operating System*) nebo systém Windows, je nazýván *command* nebo *commnad line* (Obrázek 1).



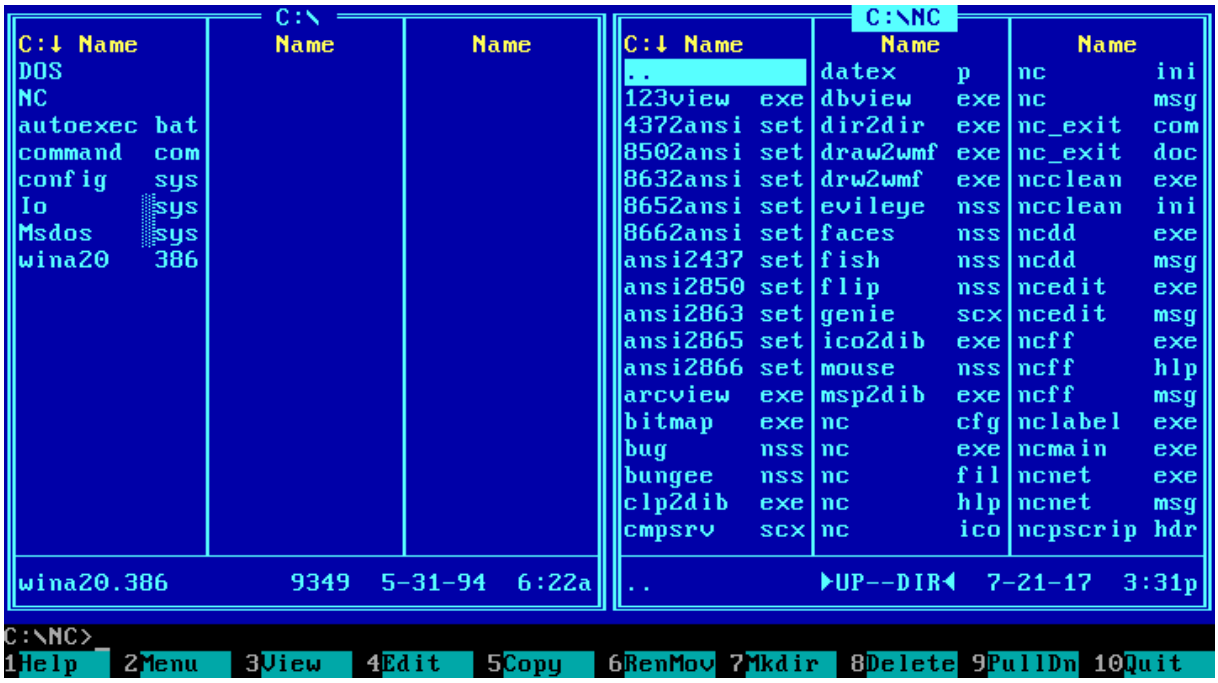
Obrázek 1 - Příkazová řádka v systému Windows

Funkce příkazového řádku byly později rozšířeny na podporu grafického režimu. Nejedná se již ale o CLI, jedná se o uživatelské rozhraní TUI, které bude popsáno v následující kapitole.

2.1.2 TUI (Text User Interface)

Textové uživatelské rozhraní (*TUI*) je rozšíření pro příkazovou řádku, kdy vstupně výstupní operace mohou být formátovány a barevně rozlišeny. Při tomto rozhraní lze využít nadstandardních funkcí příkazové řádky, jako je zobrazení okna poskládaného ze speciálních znaků a tím vytvořený dojem, že se jedná o grafickou aplikaci. Pohyb v těchto aplikacích probíhá ve většině případů pomocí kurzorových šipek. Není vyloučena i přítomnost ukazatele neboli myši.

Příkladem může být program *Norton Commander* (Obrázek 2), správce souborů pod systémem DOS. Tato aplikace obsahuje tabulku se seznamem názvů souborů, která má ohraničení složené ze speciálních znaků a barevné zvýraznění. Zde je například možnost zobrazení menu, okna, tabulky, tlačítka, přepínače nebo i zaškrtávacího pole. Ty jsou složeny ze znaků a jako celek vytváří dojem grafických prvků.



Obrázek 2 - Norton Commander spuštěný v příkazové řádce [2]

2.1.3 GUI (Graphics User Interface)

Grafické uživatelské rozhraní je v dnešní době nejpoužívanějším rozhraním napříč všemi operačními systémy od unixových, přes systémy společnosti Microsoft a Apple, až po mobilní a embedded systémy.

Jedná se o uživatelské rozhraní, které obsahuje různé ovládací prvky jako jsou okna, ikony, tlačítka, posuvníky, editační řádky, pole a mnoho dalších. Tento termín byl použit v sedmdesátých letech minulého století k rozlišení grafického od textového rozhraní (příkazová řádka). První komerční GUI, které bylo pojmenováno PARC, bylo vyvinuto společností Xerox jako informační systém Xerox 8010. [3]

Toto rozhraní je primárně určeno pro ovládání pomocí myši a klávesnice (Obrázek 3), které je dostačující pro použití na stolních počítačích a noteboocích. Setkáváme se i s grafickým rozhraním, které není vhodné pro ovládání pomocí těchto vstupních zařízení, ale výhradně použitím prstů neboli tzv. dotyků. Příkladem může být operační systém Android a jeho využití v mobilních telefonech nebo tabletech.

Obrázek 3 - Grafické uživatelské rozhraní systému Microsoft Windows 7 [4]

2.2 TUI – hlavní část

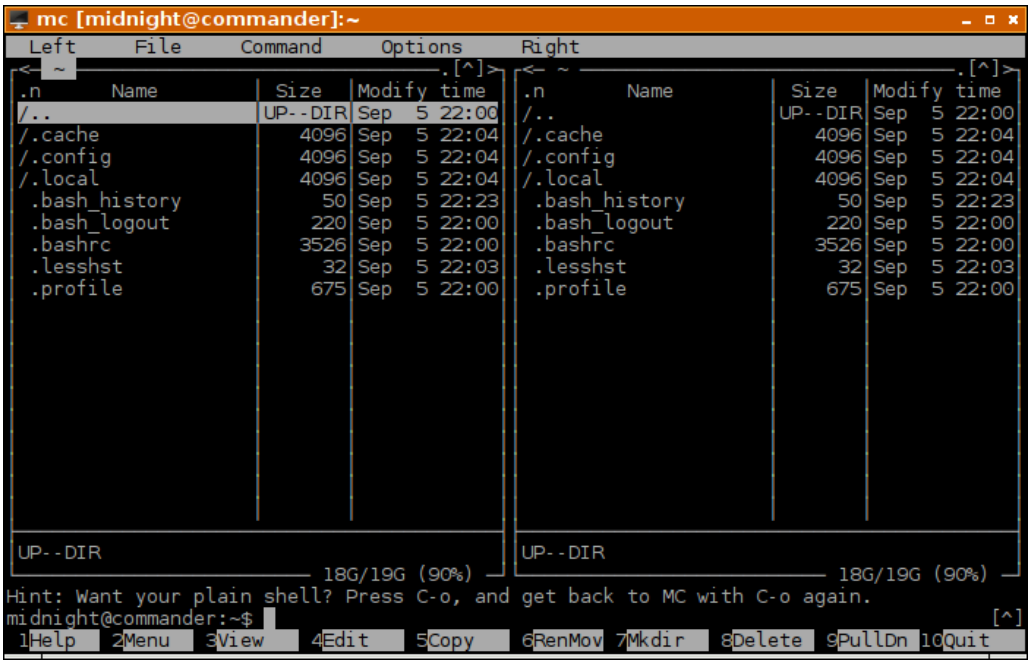
Na rozdíl od jednoduššího uživatelského rozhraní CLI je vývoj aplikací pod příkazovou řádkou využívající textové uživatelské rozhraní (*TUI*) problematičtější. Programátor musí zajistit správné vykreslení ovládacího prvku do konzole s ohledem na rozdílné rozměry konzole (rastr), tzn. rozdílný počet sloupců a řádků představující celou obrazovku.

Další problém, na který programátor musí myslet, je přepínání, zvýraznění a ovládání daného prvku v konzoli. Tento problém může vyřešit použití některé z dostupných knihoven nebo frameworku pro určitý operační systém. Příkladem může být uvedena knihovna *Ncurses* pro unixové systémy nebo knihovna *PDCurses* pro systémy MS DOS a Microsoft Windows.

2.2.1 TUI v prostředí systémů typu Unix a Linux

Jak již bylo zmíněno pro usnadnění vývoje aplikací pod příkazovým řádkem s využitím TUI se využívají různé frameworky nebo knihovny. Známá knihovna, která se hojně využívá v linuxových systémech je *Ncurses*, vycházející ze své starší verze *Pcurses*.

Knihovnu *Pcurses* napsal Pavel Curtis okolo roku 1982 a nadále byla udržována různými programátory až do roku 1986. Později byla tato verze vylepšena o několik nových funkcí, byly odstraněny některé chyby a znovu byla vydána jako *Ncurses* ve verzi 1.8.1 na konci roku 1993 Zeydem Ben-Halimem. [5]

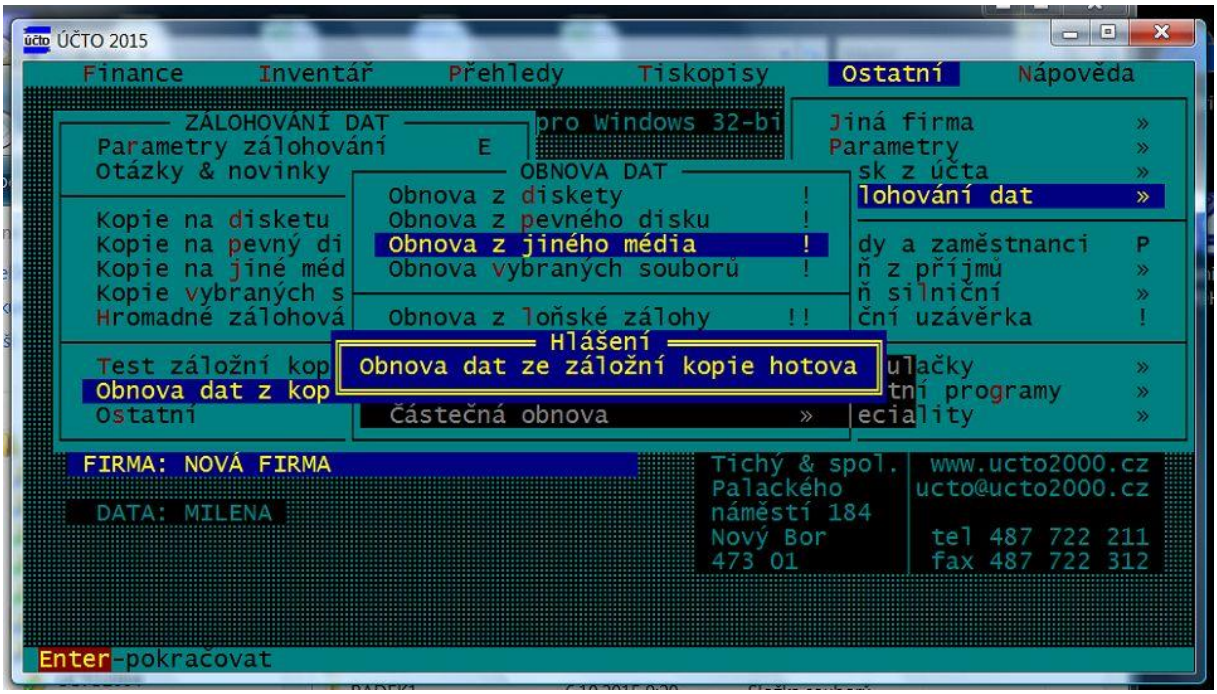


Obrázek 4 - Program Midnight Commander v příkazové řádce pod systémem typu UNIX využívající knihovnu Ncurses [6]

Aplikací využívajících knihovnu *Ncurses* v systémech linux je mnoho. Mezi nejznámější aplikace patří například souborový manažer *Midnight Commander* (Obrázek 4) nebo textový editor *vim*.

2.2.2 TUI v prostředí systému Windows

Textové rozhraní se v systémech Windows využívá i v dnešní době, kdy se setkáváme s programy běžícími v příkazové řádce, které využívají TUI. Příkladem může být program *ÚČTO* od softwarové firmy Tichý a spol., který je možné vidět na Obrázku 5.



Obrázek 5 - Program využívající TUI pod operačním systémem Windows [7]

Pro vývoj softwaru s využitím textového rozhraní je k dispozici několik frameworků nebo knihoven. Základní knihovnou je využití přímých funkcí systému, tzv. Windows API, které jsou definovány v dynamicky linkovaných knihovnách. Tato knihovna je nativní vůči jakýmkoliv programovacím jazykům. Další možností je použití knihovny *PDCurses* nebo využití vývojového prostředí určeného výhradně k tvorbě aplikací v příkazové řádce využívající TUI. Z takových prostředí stojí za zmínku *Turbo Pascal*. Tvorba aplikace pod tímto prostředím byla velice jednoduchá, jelikož se jednalo pouze o *drag & drop* a následné doprogramování funkce ovládacího prvku.

2.3 Základní možnosti konzole

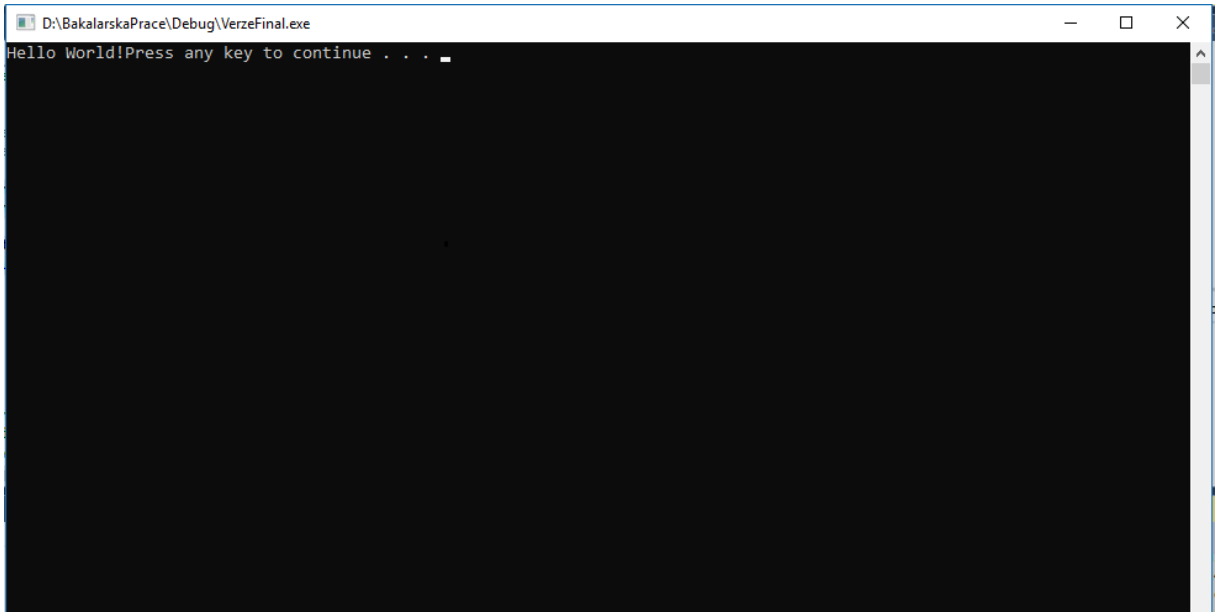
Standardní využití konzole není příliš obsáhlé. Její funkce, které jsou omezené na minimum, lze použít výhradně jen k zobrazení informací, získání dat od uživatele anebo pro základní formátování textu, jako jsou například odrážky tabulátorem nebo odřádkování. Samotná konzole nenabízí bez použití Windows API nic nadstandardního. Je to například změna barvy pozadí konzole nebo písma, zachytávání stisknutých kláves anebo nastavení kurzoru v konzoli, které je použito v praktické části této práce.

Následuje jednoduchá ukázka zdrojového kódu (Zdrojový kód 1) a obrázek výstupu na konzoli (Obrázek 6).

```
int main(void) {
	printf("Hello World!");


	return 0;
}
```

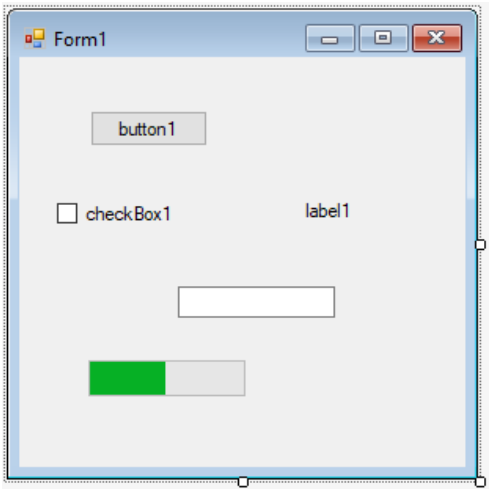
Zdrojový kód 1 - Program bez využití API



Obrázek 6 - Výsledná aplikace bez využití TUI

2.4 Windows API

Windows Application Programming Interface, dále jen Windows API je rozhraní, které umožňuje vývoj aplikací pro operační systém Windows. Toto rozhraní neobsahuje jen základní funkce jako je přístup k souborovému systému, k procesům, k pokročilým službám (např. práce s registrem, vypnutí/zapnutí systému), ale i funkce pro síťové služby, vytváření uživatelského rozhraní (např. tlačítka, menu nebo posuvníky – viz Obrázek 7) a další.



Obrázek 7 - Ukázka komponent v GUI

Windows API je možné použít pro vývoj aplikací ve všech verzích Windows. Toto rozhraní se vyskytovalo už v 16-bitových Windows a nazývalo se Win32 API. Rozdíly použití tohoto rozhraní závisí na možnostech jednotlivých verzí operačního systému. [8]

Windows API je primárně určeno pro vývoj aplikací v jazycích C nebo C++. Lze jej použít i v jiných programovacích jazycích s využitím DLL (*Dynamic link library*), ve kterých se nachází zkompilované funkce.

Windows API je nedílnou součástí pro vývoj aplikací v konzoli pod systémem Windows.

2.5 Rozšířená práce s konzolí

Součástí programovacího jazyka C++ pod operačním systémem Microsoft Windows je knihovna *Windows.h*, která zpřístupní funkce pro rozšířenou práci s konzolí. Jedná se o hlavičkový soubor, který obsahuje deklaraci všech funkcí z Windows API, maker a datových typů, které jsou využívány napříč různými funkcemi.

Framework využívá jen malou část z této knihovny, které jsou například pro potřebu zachycení stisknuté klávesy, nastavení aktuální barvy konzole nebo nastavení pozice a viditelnosti kurzoru. Přehled použitých funkcí ve frameworku:

- *GetStdHandle()* – získání odkazu na okno konzole,
- *SetConsoleCursorPosition()* – nastavení pozice kurzoru,
- *GetAsyncKeyState()* – získání informace o konkrétní stisknuté klávese,
- *CreateConsoleScreenBuffer()* – vytvoření mezipaměti pro rychlejší vykreslení,
- *SetConsoleActiveScreenBuffer()* – nastavení mezipaměti,
- *WriteConsoleOutput()* – vykreslení požadovaného obrazu do konzole,
- *SetConsoleCursorInfo()* – nastavení viditelnosti kurzoru,
- *SetConsoleTextAttribute()* – nastavení konkrétní barvy.

Mezi další funkce, které nebyly použity ve frameworku mohu zmínit například:

- *SetConsoleTitleA()/SetConsoleTitleW()* – nastavení titulku okna konzole,
- *ReadConsoleOutput()* – čtení obsahu obrazovky konzole.

Základním kamenem pro rozšířenou práci s konzolí je datový typ *HANDLE*, který obsahuje odkaz na objekt, v tomto případě na okno konzole. K získání odkazu na konzolové okno slouží funkce *GetStdHandle()* jejímž parametrem je informace, zda chceme získat handle pro vstupní nebo výstupní operace. Jelikož se jedná pouze o zobrazení v konzoli, handle je získán s parametrem *STD_OUTPUT_HANDLE*. Jedná se o makro, které je definováno v hlavičkovém souboru *WinBase.h* zpřístupněné přes knihovnu *Windows.h*.

Díky odkazu na okno konzole je nejen možné získat informace, ale také nastavit velikost okna konzole, barvu, pozici kurzoru a mnoho dalšího.

Framework získává informace přes *HANDLE* o konzoli okna do datových struktur, které jsou též definovány v knihovně, jako je *CONSOLE_SCREEN_BUFFER_INFO* pro rozměry okna a *CONSOLE_CURSOR_INFO* pro informaci o kurzoru v konzoli.

2.6 Barvy v konzoli

Barvy v konzoli jsou omezené na velikost 1 bajt neboli 16 typů barev pro pozadí (konzole) a na 16 typů barev pro popředí (text). Počet kombinací je 256 a tudíž hodnota barvy nabývá této velikosti. Viz následující tabulka (Tabulka 1).

Tabulka 1 - Přehled možných barev v konzoli

Název barvy ve frameworku pro pozadí (konzole)	Hodnota	Název barvy ve frameworku pro popředí (text)	Hodnota
`ColorsBackground::BLACK`	`0x00`	`ColorsForeground::BLACK`	`0x00`
`ColorsBackground::DARK_BLUE`	`0x10`	`ColorsForeground::DARK_BLUE`	`0x01`
`ColorsBackground::DARK_GREEN`	`0x20`	`ColorsForeground::DARK_GREEN`	`0x02`
`ColorsBackground::DARK_CYAN`	`0x30`	`ColorsForeground::DARK_CYAN`	`0x03`
`ColorsBackground::DARK_RED`	`0x40`	`ColorsForeground::DARK_RED`	`0x04`
`ColorsBackground::DARK_MAGENTA`	`0x50`	`ColorsForeground::DARK_MAGENTA`	`0x05`
`ColorsBackground::DARK_YELLOW`	`0x60`	`ColorsForeground::DARK_YELLOW`	`0x06`
`ColorsBackground::SILVER`	`0x70`	`ColorsForeground::SILVER`	`0x07`
`ColorsBackground::GRAY`	`0x80`	`ColorsForeground::GRAY`	`0x08`
`ColorsBackground::BLUE`	`0x90`	`ColorsForeground::BLUE`	`0x09`
`ColorsBackground::GREEN`	`0xA0`	`ColorsForeground::GREEN`	`0x0A`
`ColorsBackground::CYAN`	`0xB0`	`ColorsForeground::CYAN`	`0x0B`
`ColorsBackground::RED`	`0xC0`	`ColorsForeground::RED`	`0x0C`
`ColorsBackground::MAGENTA`	`0xD0`	`ColorsForeground::MAGENTA`	`0x0D`
`ColorsBackground::YELLOW`	`0xE0`	`ColorsForeground::YELLOW`	`0x0E`
`ColorsBackground::WHITE`	`0xF0`	`ColorsForeground::WHITE`	`0x0F`

2.7 Kurzor v konzoli

Windows API umožňuje nastavení velikosti a pozice kurzoru. Pro nastavení velikosti je využívána datová struktura CONSOLE_CURSOR_INFO. Přes instanci této struktury a použití metody z knihovny *SetConsoleCursorInfo()* je nastaven kurzor na neviditelný a během vykreslení je zamezeno blikání kurzoru po celé ploše konzole. Použitím metody *WriteConsoleOutput()* se zamezilo blikající obrazovce, která se překreslovala.

Použití metody *SetConsoleCursorPosition()* pro nastavení pozice kurzoru je využito v metodě zajišťující vykreslení komponenty *EditBox*, u které je zadávání textu primární funkcí. U tohoto prvku je obnovena i viditelnost kurzoru.

Následuje ukázka (Zdrojový kód 2), jak lze pomocí Windows API nastavit pozici a viditelnost kurzoru v konzoli.

```
// Set visible cursor
CONSOLE_CURSOR_INFO info;
info.bVisible = false;
info.dwSize = 100;
SetConsoleCursorInfo(this->hStdOut, &info);
```

Zdrojový kód 2 - Nastavení viditelnosti kurzoru

2.8 Zachycení stisknuté klávesy

Ve Windows API je k dispozici funkce *GetAsyncKeyState()*, jak již bylo zmíněno v předchozí kapitole, která vrací hodnotu *boolean* podle toho, zda daná klávesa byla stisknuta. Jejím parametrem je celé číslo, což představuje právě konkrétní klávesu. V následující tabulce (Tabulka 2) jsou vypsány hodnoty stěžejních kláves, které byly použity v této práci. [9]

Tabulka 2 - Použité konstanty kláves ve frameworku

Název konstanty (hodnota)	*Klávesa*
VK_TAB (0x09)	Klávesa tabulátor
VK_RETURN (0x0D)	Klávesa enter
VK_SPACE (0x20)	Klávesa mezerník
VK_ESCAPE (0x1B)	Klávesa escape
VK_UP (0x26)	Klávesa šipka nahoru
VK_DOWN (0x28)	Klávesa šipka dolů
VK_LEFT (0x25)	Klávesa šipka doleva
VK_RIGHT (0x27)	Klávesa šipka doprava

3 Návrh a implementace frameworku

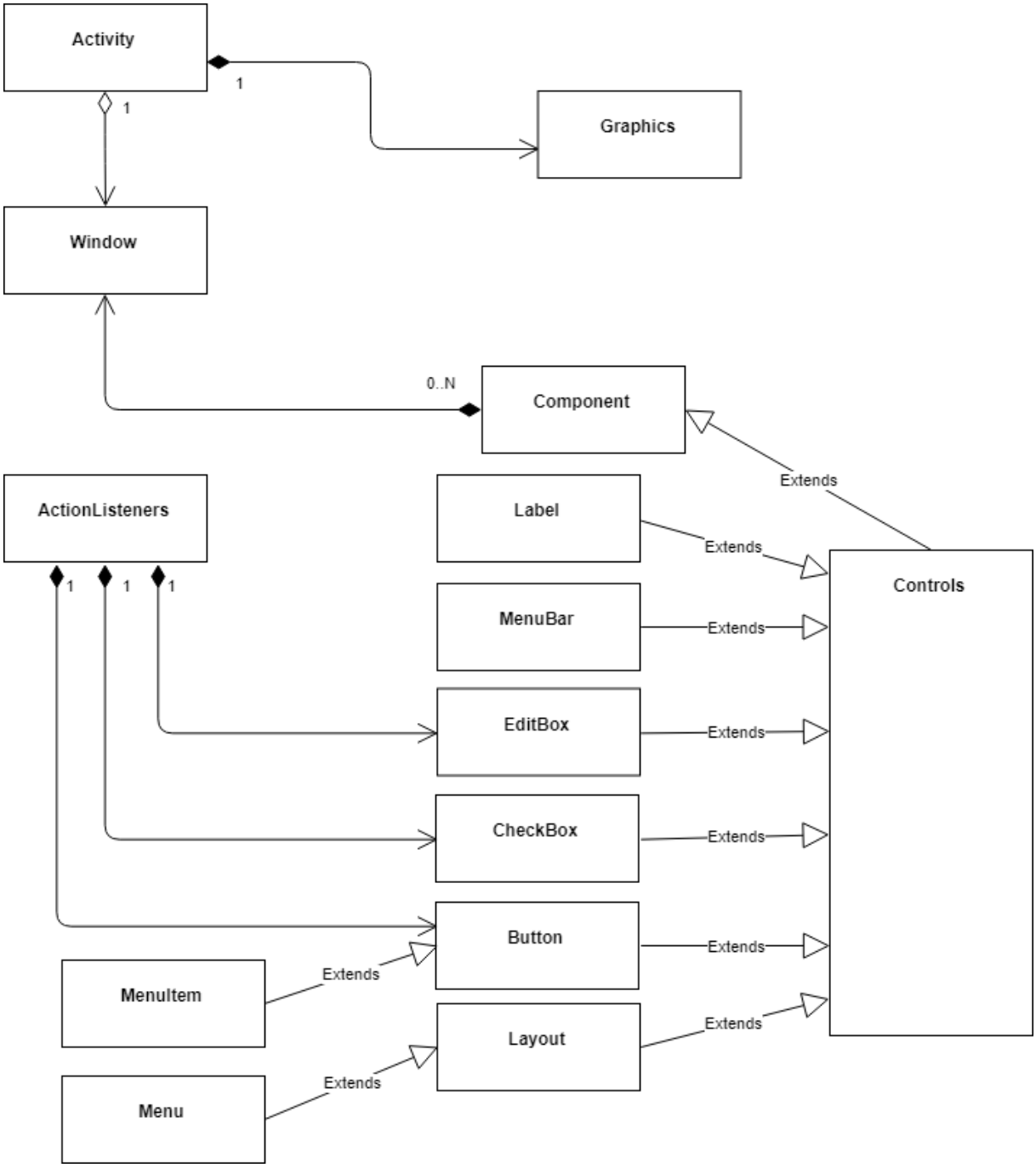
Framework byl navržen v co nejjednodušším provedení. Obsahuje několik tříd, které jsou nezbytné pro jeho správnou funkčnost. Využívá také standardní knihovnu šablon (*Standard Template Library*, zkráceně STL), konkrétně kontejner typu *list*, a některé příslušné algoritmy dostupné v knihovně *algorithm*.

Základ tohoto frameworku tvoří třída *Activity*, která zajišťuje přímé vykreslování do konzole nebo správu předem vytvořených objektů neboli oken (*Window*). Každé okno se nadále chová jako kontejner pro komponenty, které jsou součástí frameworku. Komponenty neboli ovládací prvky, které jsou v tomto frameworku k dispozici:

- *Label* – statický text,
- *Button* – tlačítko,
- *EditBox* – textové pole pro zadávání textu,
- *CheckBox* – zaškrtávací pole,
- *MenuBar* – hlavní (horní) lišta menu,
- *MenuItem* – položka menu.

Dále framework obsahuje složený objekt pro zobrazení zprávy, tzv. *MessageBox*.

3.1 UML diagram tříd frameworku



Podrobnější diagram, který obsahuje předpis atributů a metod jednotlivých tříd je k dispozici v přílohách (Příloha A).

3.2 Základní třídy

3.2.1 Třída Activity

Základním prvkem tohoto frameworku je třída *Activity*. Tato třída obsahuje metody pro přidání nebo odebrání objektu typu *Window*, metodu pro vykreslení všech objektů, získání instance třídy *Graphics* a metodu pro ukončení všech zobrazených oken a ukončení životního cyklu frameworku.

3.2.2 Třída Graphics

Graphics je tzv. vykreslovacím objektem do konzole. Nese informaci v každém poli o barvě konzole, písma a znaku. Obsahuje metodu *AddChar()* pro přidání znaku a barvy na konkrétní pozici, metodu pro vykreslení do konzole *Repaint()*, metodu pro vyčištění konzole *Clear()*, funkce k získání šířky *GetScreenWidth()* a výšky *GetScreenHeight()* konzole a funkci na zjištění změny pro vykreslení *IsChange()*. Tento stav se mění během volání metody *AddChar()* a metody *Repaint()*.

Následuje ukázka (Zdrojový kód 3), jak probíhá vykreslení již sestaveného obrazu do konzole s využitím Windows API a funkce *WriteConsoleOutput()* pro vykreslení mapy obsahující informace o znaku a barvě na každé pozici.

```
void Framework::Graphics::Repaint()
{
	if (!this->change) return;

	WriteConsoleOutputW(
		this->hStdOut,
		this->chiBuffer,
		this->sizeRectBuffer,
		this->positionZero,
		&this->srctScreen
	);

	this->change = false;
}
```

Zdrojový kód 3 - Vykreslení obrazu do konzole s využitím Windows API a bufferu

Původní verze frameworku využívala standardní cyklus. Tato metoda nebyla vyhovující z důvodu jejího pomalého vykreslování.

Třída *Graphics* využívá možnosti obrazové mezipaměti, která je o poznání rychlejší oproti klasickému cyklu. Informace o znaku a použité barvě je uchovávána v poli o velikosti rastru

obrazovky konzole. Struktura je datového typu *CHAR_INFO* definovaná v knihovně *Windows.h*.

3.2.3 Třída Window

Window je základní objekt, vycházející ze třídy *Component*, na který se umisťují kontrolky (ovládací prvky) pomocí metody *AddControl()* a je zajištěno jejich ovládání pomocí kláves. Tento objekt se dá nazvat i jako kontejner. Jako ukázka vytvoření objektu typu *Window* a zaregistrování do instance třídy *Activity* slouží Zdrojový kód 4.

```
mainAct = new Activity();
window = new Window(mainAct);
Button *btnKonec = new Button(L"KONEC");
btnKonec->SetPosition(8, 11);
btnKonec->OnAction += closeApp;
window.AddControl(btnKonec);
```

Zdrojový kód 4 - Použití objektu typu Window

Každý objekt typu *Window* má svůj životní cyklus, který je tvořen vláknem. Toto vlákno získává informaci o stisknutých klávesách a předává je podle potřeby ovládacím prvkům, které jsou v tomto kontejneru obsaženy. Následně, pokud byla zachycena událost stisknuté některé ze stěžejních kláves, je volána metoda pro překreslení. Vlákno je inicializováno po zavolání metody *Show()*. Dále třída *Window* obsahuje metody *Hide()* pro skrytí kontejneru, metody a funkce pro nastavení nebo zjištění stavu (např. režim okna Fullscreen, Center) nebo typ orámování.

3.2.4 Třída Component

Jedná se o abstraktní třídu, která obsahuje dvě virtuální metody a to pro vykreslení *Paint()* a zachycení stisknutých kláves *PressKey()*. Zároveň obsahuje základní proměnné pro nastavení nebo získání pozice, rozměru nebo i viditelnosti komponenty.

Tyto dvě metody jsou v každém ovládacím prvku individuálně implementovány v závislosti na konkrétním prvku a na tom, co má představovat.

3.2.5 Třída Control

Control vychází z třídy *Component* a je výchozím objektem pro vytvořené ovládací prvky, které byly v tomto frameworku předpřipraveny. Tato třída je rozšířena o metodu a funkci pro

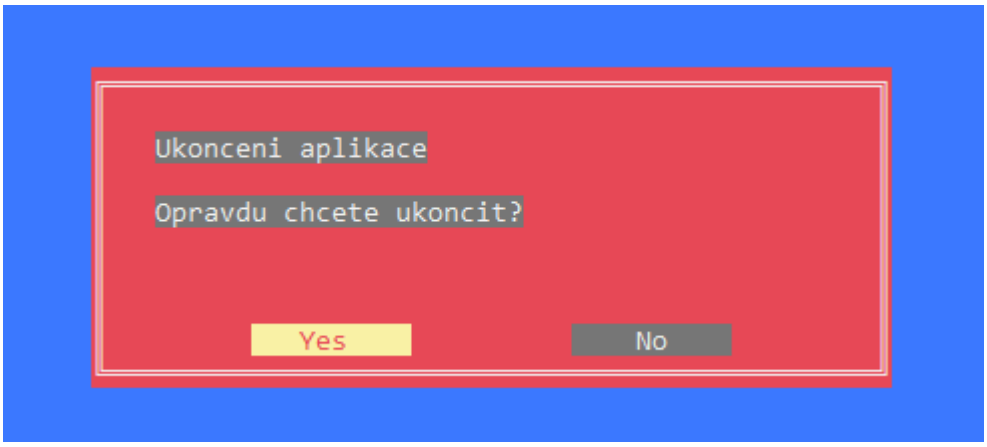
nastavení anebo získání hodnoty pro barvu, která je použita pro zvýraznění aktuálně vybraného ovládacího prvku.

3.3 Controls

Součástí frameworku jsou i předdefinované základní ovládací prvky, které vycházejí ze třídy *Control*. Veškeré kontrolky jsou přístupné ze jmenného prostoru *Framework::Controls*. Pro uvedení jednoduchých příkladů byly vytvořeny tyto ovládací prvky:

- *Button* – klasické tlačítko, které může obsahovat metodu pro akci (*OnAction*),
- *Label* – statický text,
- *CheckBox* – zaškrtávací tlačítko, které může obsahovat metodu pro změnu stavu (*OnChange*),
- *EditBox* – vstupní zadávací pole pro text a možnost nastavení metody pro změnu stavu komponenty (*OnChange*),
- *MenuBar* – hlavní menu,
- *MenuItem* – položka hlavního menu (*MenuBar*) nebo podmenu (*Menu*), která vychází ze třídy *Button*,
- *Menu* – podmenu, které se přiděluje k určité položce menu (*MenuItem*).

Dále framework nabízí, jak již bylo zmíněno, statickou metodu *MsgBox()*. Slouží pro zobrazení dialogu (viz Obrázek 8). Metoda má vstupní parametry, kterými jsou ukazatel na instanci třídy *Activity*, text hlavičky, text zprávy a druhy tlačítek.



Obrázek 8 - Výsledek použití metody MessageBox

Tato metoda pak vrací typ návratového tlačítka, tzv. *DialogResult*, kterým byl dialog ukončen. Tento návratový typ vrací následující stavy:

- BTN_OK,
- BTN_YES,
- BTN_NO,
- BTN_CANCEL,
- BTN_ABORT,
- BTN_RETRY,
- BTN_IGNORE,
- NONE.

Inspirace ke kombinaci tlačítek byla převzata z jazyka C#[1], kde je tento dialog často využíván při vývoji klasických desktopových aplikací. Využívá se především pro zobrazení zprávy informačního charakteru nebo dotazovací jako je například změna dat a jejich uložení.

Dále mohou některé z ovládacích prvků obsahovat obslužnou událost (metodu) pro vykonání akce. Například pro tlačítko (*Button*) je k dispozici metoda *OnAction*, pro zaškrtávací pole (*CheckBox*) je to metoda *OnChange*. Komponenty, které obsahují událost, jsou typu *ActionListeners*. Základní tvar metody má jeden parametr, a to ukazatel na objekt typu *Component* a návratový datový typ *void*. Tento parametr obsahuje ukazatel na objekt, který volá tuto metodu.

Následující Zdrojový kód 5 znázorňuje implementaci vlastní události a asociace s komponentou.

[1] Více informací na https://docs.microsoft.com/en-us/dotnet/api/system.windows.forms.messageboxbuttons?view=netframework-4.7.2

```
Button* bExit;
Activity* mainAct;

void closeApp(Component* c) {
	// Tělo metody
	if (MsgBox(mainAct, L"Ukonceni aplikace", L"Opravdu chcete ukoncit?",
		MessageBoxBtns::YesNo) == DialogResult::BTN_YES)
	{
		mainAct->Exit(nullptr);
	}
}

int main()
{
	bExit = new Button(L"Ukončit");
	bExit->SetPosition(5, 3);
	bExit->OnAction += closeApp;
}
```

Zdrojový kód 5 - Implementace metody a následná asociace s komponentou

V příkladu lze vidět tlačítko, které je asociováno s obslužnou metodou, tak aby tlačítko vykonávalo příslušnou operaci.

3.3.1 Button (tlačítko)

Komponenta typu *Button* představuje funkci klasického tlačítka. Obsahuje metody pro správu atributů jako je text, barva pozadí/popředí. Dále obsahuje operátor += pro přidání události *OnAction*, která se po stisku klávesy Enter zavolá.

3.3.2 Label (statický text)

Label je statický text, který může sloužit jako popisek, text apod. Obsahuje pouze metody pro nastavení barvy pozadí/popředí a textu.

3.3.3 CheckBox (zaškrtávací pole)

Komponenta typu *CheckBox* slouží pro grafické vyjádření hodnoty Ano/Ne. Dále obsahuje totožné metody jako předchozí komponenta *Label* a obslužnou metodu *OnChange*. Ta je volána při každé změně hodnoty z Ano na Ne a opačně. Na Obrázku 9 je znázorněno vykreslení této komponenty.

Obrázek 9 - Náhled komponenty CheckBox

3.3.4 EditBox (textové pole)

EditBox je vstupní pole, které v sobě udržuje text zadaný uživatelem. Změna, smazání nebo vytvoření textu v poli je voláno klávesou Enter. Obsahuje metodu *OnChange*, která je totožná s metodou u komponenty *CheckBox.*

3.4 Ukázka použití frameworku v praxi

Pro použití frameworku je důležité v prvé řadě připojit hlavičkový soubor *Framework.h*. Ke správnému použití je důležité vytvořit jednu instanci objektu *Activity*. Tento objekt je výchozí pro zavedení frameworku, který zajišťuje překreslování obrazovky konzole. Viz Zdrojový kód 6.

```
#include "Framework.h"

using namespace Framework;

Activity* mainAct;

int main()
{
	mainAct = new Activity();

	return 0;
}
```

Zdrojový kód 6 - Vytvoření instance třídy Activity

Dalším krokem je vytvoření alespoň jedné instance třídy *Window*. Tento objekt se registruje přes metodu *AddWindow()*, která je implementovaná ve třídě *Activity*.

Správná funkčnost tohoto frameworku je závislá nejen na zaregistrovaných objektech typu *Window*, ale i na jejich viditelnosti. Ta je nastavena metodou *Show()* pro zobrazení a metodou *Hide()* pro zneviditelnění. Pokud existuje odkaz v instanci třídy *Activity* alespoň na jeden viditelný objekt typu *Window*, framework bude aplikován a jeho životní cyklus potrvá do ukončení neboli zneviditelnění okna. Třída *Activity* nabízí metodu *Exit()* pro hromadné ukončení všech registrovaných a zároveň zviditelněných formulářů v dané instanci.

```
window = new Window(mainAct);
window->Show();
```

Zdrojový kód 7 - Vytvoření a zobrazení okna

Na Zdrojovém kódu 7 je znázorněn příklad asociace okna s hlavní aktivitou. Tato asociace probíhá v konstruktoru objektu. Voláním metody *Show()* se okno zviditelní a vytvoří se vlákno. To se opakuje do doby, než okno změní stav viditelný na neviditelný.

Nyní je framework ve stavu připraven a funkční. Pokud se aplikace spustí, zobrazí se pouze černé okno konzole. To znamená, že výstup je v pořádku. Ověřit to lze tím, že se změní barva pozadí objektu *Window*, jeho pozice a rozměry nebo nastavení plného obrazu, viz Zdrojový kód 8.

```
// Změna barvy pozadí a popředí
w->SetColor(ColorsBackground::BLUE, ColorsForeground::WHITE);

// Nastavení plného obrazu
w->SetFullscreen(true);

// Nastavení rozměru a pozice
w->SetX(10);
w->SetY(10);
w->SetWidth(20);
w->SetHeight(5);
```

Zdrojový kód 8 – Změny barvy pozadí, zapnutí plného obrazu a nastavení pozice a rozměrů okna

Dalším krokem je správné rozvrhnutí aplikace, tzn. jak bude vypadat a co bude obsahovat.

Aby bylo možné aplikaci ukončit bez jakýchkoliv problémů, je nutné vytvořit tlačítko s obslužnou událostí, která zajistí její ukončení. V prvé řadě je vytvoření instance objektu typu *Button*. Dále je zapotřebí nadefinovat metodu a asociovat ji s tlačítkem. Do těla metody se vloží volání metody, která zajistí ukončení aplikace. Ta je součástí instance objektu *Activity*, jak již bylo zmíněno v této práci. Viz Zdrojový kód 9.

```
Button* bExit;
Activity* mainAct;

void closeApp(Component* c) {
	mainAct->Exit(nullptr);
}

int main()
{
	bExit = new Button(L"Ukončit");
	bExit->SetPosition(5, 3);
	bExit->OnAction += closeApp;

	w->AddControl(bExit);
}
```

Zdrojový kód 9 - Vytvoření ukončovacího tlačítka

Použití komponenty typu menu se může zdát komplikované, ale bylo navrženo co nejjednodušeji. Pokud je jeho použití v aplikaci nezbytné, bude jeho vytvoření spočívat v kombinaci objektů typu *MenuBar* (hlavní lišta) a *MenuItem* (položkách seznamu). Na Zdrojovém kódu 10 je možné vidět aplikování hlavní lišty menu v praxi.

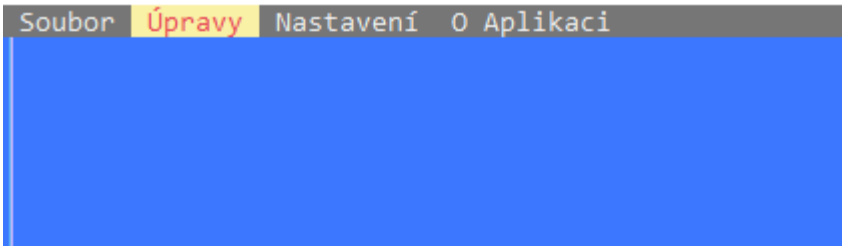
```
MenuBar* mb = new MenuBar();
MenuItem* miSoubor, *miUpravy, *miNastaveni, *miO;


miSoubor = new MenuItem(L"Soubor");
miUpravy = new MenuItem(L"Úpravy");
miNastaveni = new MenuItem(L"Nastavení");
miO = new MenuItem(L"O Aplikaci");

mb->AddMenuItem(miSoubor);
mb->AddMenuItem(miUpravy);
mb->AddMenuItem(miNastaveni);
mb->AddMenuItem(miO);
```

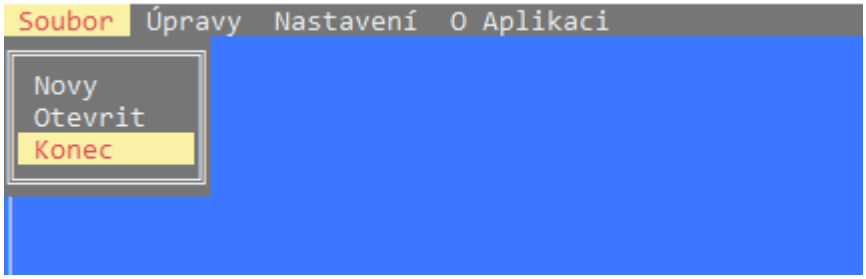
Zdrojový kód 10 - Hlavní lišta menu a její základní nabídka

První krok je vytvoření instance typu *MenuBar*. Dále instance položek *MenuItem* a jejich přiřazení k hlavní liště přes metodu *AddMenuItem()*. Popisky položek mohou obsahovat české znaky, jelikož framework využívá tzv. Unicode. Nyní lze spuštěním aplikace ověřit funkčnost menu. Viz Obrázek 10.



Obrázek 10 - Hlavní lišta menu

K vytvoření podmenu v hlavní liště lze dojít k výsledku jako je to znázorněno na Obrázku 11. K jeho vytvoření slouží kontejner typu *Menu*. Obdobně jako to je u hlavní lišty, tak i u podmenu se vytvoří objekty položky seznamu a přiřadí se do kontejneru přes metodu *AddItem()*, kde je parametrem této metody ukazatel na položku seznamu.



Obrázek 11 - Hlavní menu s nabídkou podmenu

Kontejner se poté musí asociovat s položkou v hlavní liště menu. K tomu byla použita metoda *SetSubmenu()*, kde je parametrem této funkce ukazatel na kontejner.

3.5 Unicode[2]

Tento framework, jak již bylo zmíněno, pracuje se znakovou sadou typu Unicode, aby byla zaručena kompatibilita s různými sadami národních znaků. Důvodem použití této sady jsou další latinské znaky používané v evropských zemích. [10]

„*Unicode doposud zahrnuje přes 96 000 symbolů a 49 skriptů a stále se rozvíjí.*“ [10]

Použitím sady Unicode v tomto frameworku lze zajistit jazykovou kompatibilitu aplikace. Například pro zobrazení rozšířených latinských znaků se před uvozovkami datového typu *string* používá znak L (viz Zdrojový kód 11).

```
Activity* mainAct;

MsgBox(mainAct, L"Ukončení aplikace", L"Opravdu chcete ukončit?",
MessageBoxBtns::YesNo
```

Zdrojový kód 11 - Příklad použití unicode v C++

[2] Více na www.unicode.org

3.6 Použité funkce a datové struktury Windows API

V této práci jsou použity metody Windows API zaručující rychlost a rozšířené možnosti s konzolí. Dále datové struktury, které se využívají v metodách API. Následuje seznam a popis použitých metod:

- *GetStdHandle()* – získání odkazu na zařízení (konzoli). Parametrem je hodnota typu DWORD určující, zda chceme pracovat se vstupní vyrovnávací pamětí nebo s aktivní vyrovnávací pamětí. [11]
- *WriteOutputConsole()* – zápis znaků a druh barvy do zadaného obdélníkového bloku znakových buněk ve vyrovnávací paměti obrazovky konzole. Parametrem této funkce je handle s aktivní vyrovnávací pamětí. Dále pole datové struktury CHAR_INFO, velikost pole, počáteční pozice vykreslení a posledním parametrem je rozměr plochy k zápisu. [12]
- *CreateConsoleScreenBuffer()* – metoda vracející nový handle. Parametrem jsou přístupová práva do vyrovnávací paměti, typ vyrovnávací paměti obrazovky nebo mód sdílení paměti. [13]
- *SetConsoleActiveScreenBuffer()* – nastaví handle vyrovnávací obrazové paměti na výchozí. Vstupním parametrem je handle. [14]
- *GetConsoleScreenBufferInfo()* – získá informace o vyrovnávací paměti obrazovky konzole. Parametrem je handle, který musí mít právo přístupu GENERIC_READ a ukazatel na datovou strukturu typu CONSOLE_SCREEN_BUFFER_INFO. Z této struktury je získán rozměr okna a jeho počet znaků na délku a na výšku. [15]
- *GetConsoleCursorInfo()* – získání informace o velikosti a viditelnosti kurzoru v konzoli. Parametrem je handle paměti a ukazatel na datovou strukturu *PCONSOLE_CURSOR_INFO*. [16]
- *SetConsoleCursorInfo()* – nastavení kurzoru, jeho velikost a viditelnost. Parametry této funkce jsou stejné jako u *GetConsoleCursorInfo()*. [17]
- *SetConsoleCursorPosition()* – nastavení pozice kurzoru. Parametrem jsou handle aktuální paměti a datový typ *COORD*, který obsahuje souřadnice, na které se přesune kurzor. [18]
- *GetAsyncKeyState()* – funkce určující, zda daná klávesa byla před voláním této funkce stisknuta nebo ne. Parametrem je číselná hodnota reprezentující zvolenou klávesu. Přehled stěžejních kláves, které byly v této práci použity, jsou popsány v Tabulce 2.

Jako další je uveden seznam použitých datových struktur definované v hlavičkovém souboru *Windows.h*:

- CHAR_INFO – struktura obsahující atributy o použité barvě a reprezentaci znaku v sadě ASCII nebo Unicode.

```
typedef struct _CHAR_INFO {

  union {
    WCHAR UnicodeChar;
    CHAR  AsciiChar;
  } Char;
  WORD  Attributes;
} CHAR_INFO, *PCHAR_INFO;
```

Zdrojový kód 12 - Definice struktury CHAR_INFO [19]

- CONSOLE_SCREEN_BUFFER_INFO – struktura obsahující atributy o rozměrech okna, pozice kurzoru, atributu s přístupovými právy a rozměrech vykreslovací plochy. Viz Zdrojový kód 12.

```
typedef struct _CONSOLE_SCREEN_BUFFER_INFO {
  COORD      dwSize;
  COORD      dwCursorPosition;
  WORD       wAttributes;
  SMALL_RECT srWindow;
  COORD      dwMaximumWindowSize;
} CONSOLE_SCREEN_BUFFER_INFO;
```

Zdrojový kód 13 - Definice datové struktury CONSOLE_SCREEN_BUFFER_INFO [20]

- CONSOLE_CURSOR_INFO – informace o nastaveném kurzoru ve vyrovnávací paměti.

```
typedef struct _CONSOLE_CURSOR_INFO {

  DWORD dwSize;
  BOOL  bVisible;
} CONSOLE_CURSOR_INFO, *PCONSOLE_CURSOR_INFO;
```

Zdrojový kód 14 - Definice datové struktury CONSOLE_CURSOR_INFO [21]

4 Závěr

V této práci byl vytvořen funkční framework uživatelského rozhraní v konzoli. Pro tento projekt vznikla knihovna, která usnadňuje vývoj aplikací pro příkazovou řádku v systémech Windows. Použitelnost této knihovny je možné i v jiném projektu. Nakonec byl vytvořen ukázkový program, který demonstruje využití této knihovny.

Aktuální verze knihovny disponuje se 14 třídami, 5 výčtovými typy a 1 statickou metodou o přibližném počtu 1500 řádků. Ukázkový program pak obsahuje přibližně 150 řádků.

Tento framework byl navržen tak, aby jeho použití bylo co nejjednodušší. Původní myšlenka byla vytvořit framework čistě procedurální, tzn. volání statických metod. Ale nakonec byl framework navržen tak, že používá objektový přístup, který ho dělá ještě jednodušším. Přímý přístup k objektu a změna požadovaného atributu se promítne na obrazovku ihned a bez zbytečného vyplňování dalších hodnot. Tím je myšlena verze se statickými metodami, kde by jedna metoda obsahovala několik parametrů k nastavení konkrétní komponenty jako je viditelnost, šířka, výška, pozice apod. Vizualizace těchto komponent byla inspirována aplikací běžící pod příkazovou řádkou a využívající TUI, která byla vyobrazena na Obrázku 5.

Na demonstrujícím příkladu byla ověřena funkčnost frameworku, a zároveň jeho připravenost pro plnohodnotné nasazení v praxi. Na příkladu byla využita většina komponent obsažených v této práci. Během programování a ověřování funkčnosti byly odstraněny některé nedostatky jako je zalamování textu, pokud komponenta má atribut *Text* příliš dlouhý a další.

Výsledek práce je uspokojivý i na některé nedostatky jako je víceúrovňové menu. Rozsah vyřešení tohoto problému by byl dosti komplikovaný vzhledem k jeho postupnému překreslení a posouvání po obrazovce tak, aby předchozí nabídka nebyla překreslena. V další části vývoje je plánováno rozšíření frameworku o další komponenty, jako je například *RadioButton* (tzv. přepínač), *MemoBox* (tzv. víceřádkové pole) nebo *ProgressBar* (tzv. načítací/zpracovávací lišta). V neposlední řadě vyřešení problému, který byl zde uveden.

5 POUŽITÁ LITERATURA

[1] KLIMEŠ, Cyril. *Principy výstavby počítačů a operačních systémů* [online]. [cit. 2018-11-10]. Dostupné z: https://publi.cz/books/11/12.html

[2] *UI Museum: Norton Commander 5.0* [online]. In: . 2017 [cit. 2018-11-10]. Dostupné z: https://ilyabirman.net/meanwhile/all/ui-museum-norton-commander-5-0/

[3] GUI. *The Tech Terms Computer Dictionary* [online]. [cit. 2018-11-10]. Dostupné z: https://techterms.com/definition/gui

[4] First look at Windows 7's User Interface. In: *Ars Technica* [online]. [cit. 2018-11-13]. Dostupné z: https://arstechnica.com/information-technology/2008/10/first-look-at-windows-7/

[5] NCURSES – New Curses. In: *Thomas E. Dickey's software development projects* [online]. [cit. 2018-11-10]. Dostupné z: http://invisible-island.net/ncurses/ncurses.html

[6] How to Use Midnight Commander, a Visual File Manager. In: *SSD Cloud Hosting & Linux Servers - Linode*[online]. 2017, 12 September 2017 [cit. 2018-10-24]. Dostupné z: https://www.linode.com/docs/tools-reference/tools/how-to-install-midnight-commander/

[7] GDPR – účto 2018 verze 5. In: *Jednoduché účetnictví a daňová evidence / Účto - Tichý & spol.* [online]. 2018, 10 května 2018 [cit. 2018-11-30]. Dostupné z: http://www.ucto2000.cz/UCTOINFO/rs1803.htm

[8] Overview of the Windows API. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2010, 25 March 2010 [cit. 2018-10-26]. Dostupné z: https://docs.microsoft.com/en-us/previous-versions//aa383723%28v%3dvs.85%29

[9] Virtual-Key Codes. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs*[online]. Redmond: Microsoft, 2018, 31 May 2018 [cit. 2018-10-27]. Dostupné z: https://docs.microsoft.com/en-us/windows/desktop/inputdev/virtual-key-codes

[10] PRATA, Stephen. *Mistrovství v C*. 3., aktualiz. vyd. Brno: Computer Press, 2007. Bestseller (Computer Press). ISBN 978-80-251-1749-1.

[11] GetStdHandle function. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs*[online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/getstdhandle

[12] WriteConsoleOutput function. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/writeconsoleoutput

[13] CreateConsoleScreenBuffer function. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/createconsolescreenbuffer

[14] SetConsoleActiveScreenBuffer function. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/setconsoleactivescreenbuffer

[15] GetConsoleScreenBufferInfo function. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/getconsolescreenbufferinfo

[16] GetConsoleCursorInfo function. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/getconsolecursorinfo

[17] SetConsoleCursorInfo function. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/setconsolecursorinfo

[18] SetConsoleCursorPosition function. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/setconsolecursorposition

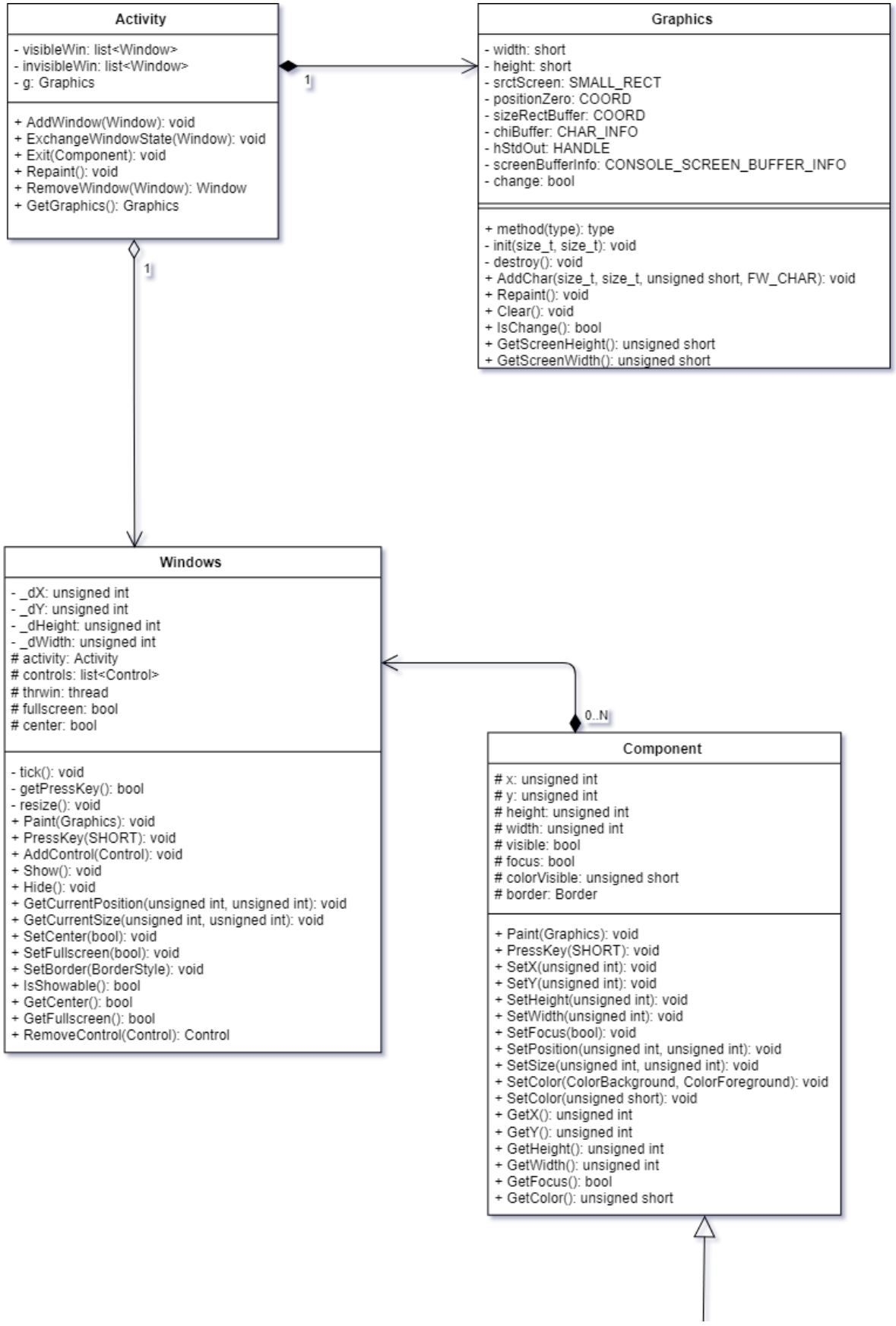
[19] CHAR_INFO structure. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs*[online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/char-info-str

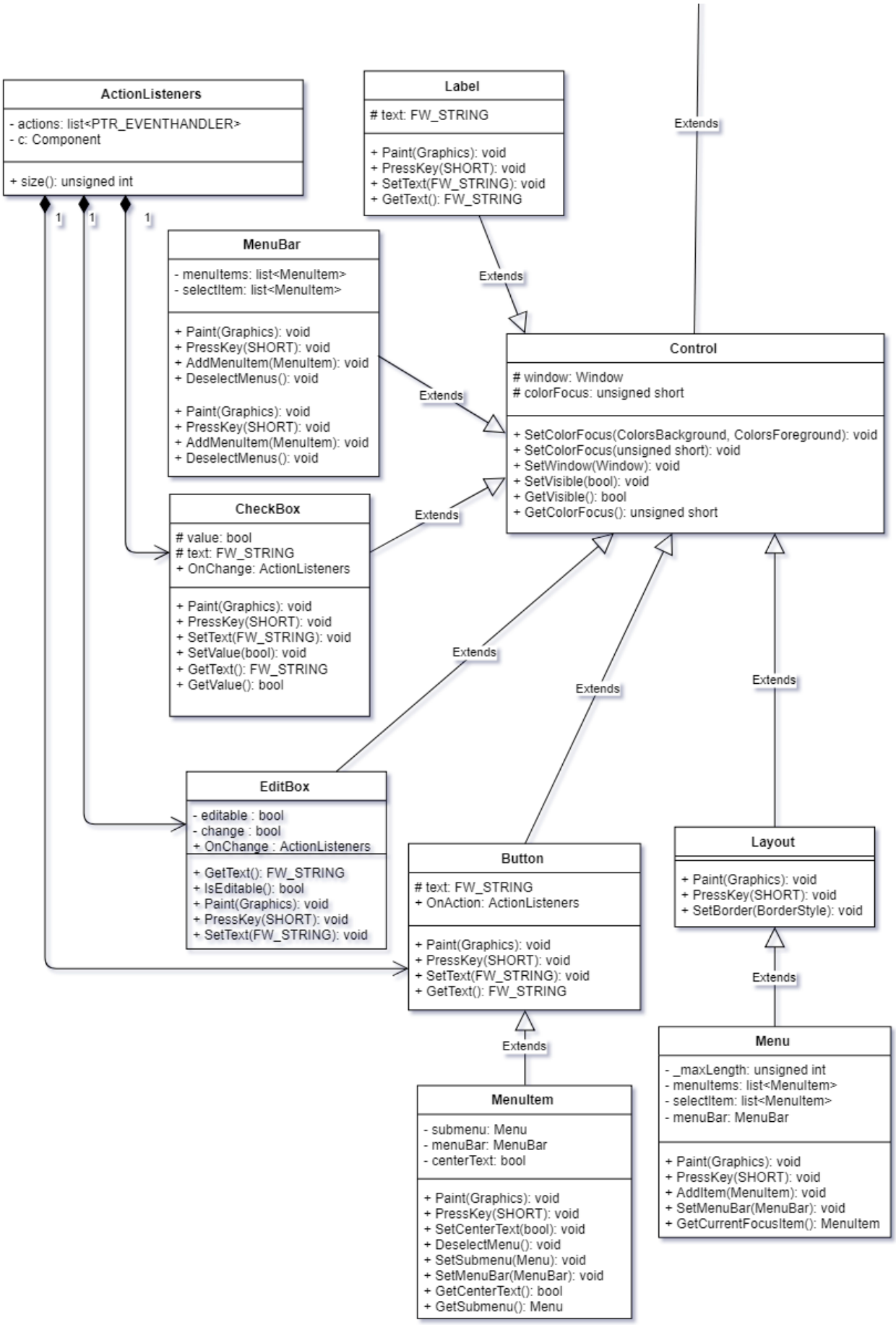
[20] CONSOLE_SCREEN_BUFFER_INFO structure. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/console-screen-buffer-info-str

[21] CONSOLE_CURSOR_INFO structure. In: *Technická dokumentace, rozhraní API a příklady kódování | Microsoft Docs* [online]. Redmond: Microsoft, 2018, 12 July 2018 [cit. 2018-11-15]. Dostupné z: https://docs.microsoft.com/en-us/windows/console/console-cursor-info-str

6 PŘÍLOHY

Příloha A – UML diagram frameworku




ActionListeners
- actions: list<PTR_EVENTHANDLER>
- c: Component
+ size(): unsigned int
Label
text: FW_STRING
+ Paint(Graphics): void
+ PressKey(SHORT): void
+ SetText(FW_STRING): void
+ GetText(): FW_STRING
Extends
1
1
1
MenuBar
- menuItems: list<MenuItem>
- selectItem: list<MenuItem>
+ Paint(Graphics): void
+ PressKey(SHORT): void
+ AddMenuItem(MenuItem): void
+ DeselectMenus(): void
+ Paint(Graphics): void
+ PressKey(SHORT): void
+ AddMenuItem(MenuItem): void
+ DeselectMenus(): void
Extends
Control
window: Window
colorFocus: unsigned short
+ SetColorFocus(ColorsBackground, ColorsForeground): void
+ SetColorFocus(unsigned short): void
+ SetWindow(Window): void
+ SetVisible(bool): void
+ GetVisible(): bool
+ GetColorFocus(): unsigned short
CheckBox
value: bool
text: FW_STRING
+ OnChange: ActionListeners
+ Paint(Graphics): void
+ PressKey(SHORT): void
+ SetText(FW_STRING): void
+ SetValue(bool): void
+ GetText(): FW_STRING
+ GetValue(): bool
Extends
Extends
Extends
Extends
Extends
EditBox
- editable : bool
- change : bool
+ OnChange : ActionListeners
+ GetText(): FW_STRING
+ IsEditable(): bool
+ Paint(Graphics): void
+ PressKey(SHORT): void
+ SetText(FW_STRING): void
Button
text: FW_STRING
+ OnAction: ActionListeners
+ Paint(Graphics): void
+ PressKey(SHORT): void
+ SetText(FW_STRING): void
+ GetText(): FW_STRING
Layout
+ Paint(Graphics): void
+ PressKey(SHORT): void
+ SetBorder(BorderStyle): void
Extends
Extends
MenuItem
- submenu: Menu
- menuBar: MenuBar
- centerText: bool
+ Paint(Graphics): void
+ PressKey(SHORT): void
+ SetCenterText(bool): void
+ DeselectMenu(): void
+ SetSubmenu(Menu): void
+ SetMenuBar(MenuBar): void
+ GetCenterText(): bool
+ GetSubmenu(): Menu
Menu
- _maxLength: unsigned int
- menuItems: list<MenuItem>
- selectItem: list<MenuItem>
- menuBar: MenuBar
+ Paint(Graphics): void
+ PressKey(SHORT): void
+ AddItem(MenuItem): void
+ SetMenuBar(MenuBar): void
+ GetCurrentFocusItem(): MenuItem