UNIVERZITA PARDUBICE

FAKULTA EKONOMICKO SPRÁVNÍ

Aplikace Microsoft Robotics pro simulaci procesů

Michal Beneš

Bakalářská práce

2011

Univerzita Pardubice
Fakulta ekonomicko-správní
Akademický rok: **2010/2011**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michal BENEŠ**
Osobní číslo:	**E08251**
Studijní program:	**B6209 Systémové inženýrství a informatika**
Studijní obor:	**Informatika ve veřejné správě**
Název tématu:	**Aplikace Microsoft Robotics pro simulaci procesů**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování:

1. Obecný popis programu
2. Popis jednotlivých části programu
3. Tvorba programu v aplikaci Microsoft Robotics

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

Sharp, John. *Microsoft Visual C 2008 : krok za krokem*. **Brno: Computer Press, 2008. ISBN-978-80-251-2027-9.**

Morgan, Sara. *Programming Microsoft Robotics Studio*. **Washington: Microsoft Press, 2008. ISBN-0-7356-2432-1.**

Brůha, Luboš. *Začínáme programovat v jazyce Visual Basic.NET*. **Praha: Computer Press, 2002. ISBN-80-7226-785-X.**

Vedoucí bakalářské práce: **Ing. Jan Panuš, Ph.D.**
Ústav systémového inženýrství a informatiky

Datum zadání bakalářské práce: **4. října 2010**

Termín odevzdání bakalářské práce: **6. května 2011**

L.S.

doc. Ing. Renáta Myšková, Ph.D.
děkanka

doc. Ing. Jiří Křupka, Ph.D.
vedoucí ústavu

V Pardubicích dne 4. října 2010

Prohlášení

Tuto práci jsem 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 mojí práci vztahují práva a povinnosti vyplívající 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 Čáslavi dne 25.dubna 2011

Michal Beneš

Poděkování

Tímto bych chtěl velmi poděkovat Ing. Janu Panušovi, Ph.D. za odbornou pomoc při tvorbě této práce.

Anotace

Práce se zabývá v první části vysvětlením pojmů algoritmus a algoritmizace. Pozornost je věnována vývojovým diagramům, pomocí kterých tyto algoritmy graficky znázornit, a dále datovým typům používaným v programování. V další části poté historii zavádění robotů do průmyslu. Závěrečná část je věnována aplikaci Microsoft Visual Programming Language, která je součástí programu Microsoft Robotics Studio. V této části jsou vysvětleny některé programy vytvořené v rámci bakalářské práce.

Klíčová slova

algoritmizace, Visual Programming Language, robot

Title

Microsoft Robotics for process simulation

Annotation

The first part of this essay considers the explanation of the terms algorithm and it's development. In this part I talk also about flow diagrams which are used to graphicly showed the algorithms. There are also mentioned the data types which are used in the programming sphere. The next part of the essay is consisted of the history of robots being used in the industry sphere for the first time. The last part considers the Microsoft Visual Programming Language application which is a part of the Microsoft Robotics Studio programme and there are explained some of the programmes created in this bachelor's essay.

Keywords

algorithm development, Visual Programming Language, robot

Obsah

Úvod

V dnešní době se tak často využívají moderní technologie, že v mnoha oborech se bez robota nelze obejít. Robot se stal v průběhu let nedílným pomocníkem v domácnostech ale především v průmyslu. Zvyšující se nároky životní úrovně obyvatelstva vedou k nutnosti zvýšení efektivity výroby. Především k tomuto účelu jsou dnes používáni průmysloví roboti. Toto však není konec vývoje, který dále směřuje k širšímu využití a zdokonalení do fáze, kdy bude robot schopen nahradit člověka a bude mu velice podobný. Díky tomuto pronikání robotů a robotiky do různých odvětví vznikají softwary pro jejich programování přístupné pro studenty a nadšence. Tyto aplikace umožňují vyzkoušet si naprogramování vlastních robotů. Některé tyto programy mají i širší možnosti využití a mohou sloužit jako výukové softwary pro začínající programátory.

V první kapitole práce bude vysvětlen pojem algoritmizace a algoritmus a jeho možný zápis pomocí vývojového diagramu. Další kapitola bude věnována samotné tvorbě algoritmu. V programování se používají určité datové typy, o kterých pojednává třetí kapitola. Čtvrtá kapitola se bude zabývat vývojem robotiky, neboť v průmyslu je dnes robot neodmyslitelný pomocník. Pátá kapitola bude pojednávat o Microsoft Robotics Studiu. Předposlední část je věnována aplikaci, která je součástí již zmíněného produktu, Microsoft Visual Programming Language. V poslední kapitole této práce jsou popsány některé programy, které byly vytvořeny v programu Microsoft Visual Programming Language.

Cílem práce je navržení programů v Microsoft Robotics Studiu, které budou sloužit pro předmět algoritmizace a programování.

1 Algoritmus, algoritmizace

V této kapitole bude vysvětlen pojem algoritmus. Nejdříve ovšem bude popsán proces, který vede ke vzniku algoritmu, algoritmizace.

Algoritmizací je myšlena činnost, jejímž vstupem je nějaký problém a výstupem algoritmus, který by měl splňovat vlastnosti uvedené níže. Při této činnosti můžeme najít více postupů vedoucích k řešení problému. [2]

S pojmem algoritmus se setkáváme již v první polovině 9. století n. l. a vyjadřoval provádění aritmetiky pomocí arabských číslic. Tehdy byl znám pod názvem algorismus. Algorismus byl odvozen od latinského jména perského matematika Abū ‘Abd Allāh Muhammad ibn Mūsā al-Khwārizmī. Později byl algorismus nahrazen slovem algoritmus. V dnešní době se s algoritmy setkáváme nejčastěji v matematice a programování. Lze ho definovat jako postup, pomocí kterého může být vyřešen zadaný problém. Podle jiné definice to je předpis vedoucí od měnitelných vstupních informací jednoznačně k požadovaným výsledkům v konečném počtu kroků.[1], [2]

Každý algoritmus má své vlastnosti, mezi které patří [2], [11], [10]:

- **Jednoznačnost (determinovanost)** - tato vlastnost zahrnuje přesnost, srozumitelnost a jednoznačnost. Každý program (algoritmus) je složen z kroků, které na sebe navzájem navazují. Tyto kroky charakterizují přechod z jednoho stavu algoritmu do jiného. V každém stavu mají data určitou formu a musí být jednoznačně určeno, jaký krok bude následovat. V rámci běžného jazyk obvykle nelze docílit této přesnosti a jednoznačnosti. Z tohoto důvodu byly pro zápis algoritmů navrženy programovací jazyky, ve kterých má každý příkaz jasně definovaný význam.
- **Konečnost (rezultativnost)** - program je konečný pokud vede k určitým výsledkům, neboli každý program musí vést k požadovaným výsledkům (řešením). Takovýto výsledek je nutné získat v "rozumném" čase a konečném počtu kroků. Za takovýto čas lze považovat dobu, po kterou má pro nás výpočet nějakou cenu**.**
- **Obecnost (hromadnost)** - program by měl být schopný řešit zadanou úlohu nejen pro jedna vstupní data. Obecnost je tedy schopnost programu řešit zadanou úlohu

s různými přípustnými vstupními daty. Příkladem je sčítání kdy program musí být schopen sčítat libovolná čísla.

- **Opakovatelnost** - tato vlastnost znamená, že algoritmus musí při opakovaném zadání stejných hodnot dospět ke stejnému výsledku jako ve všech předchozích případech.

V praxi se dále doporučuje, aby byl algoritmus srozumitelný a přehledný pro snadnější opravy a úpravy. Algoritmus pak může být vyjádřen následujícími způsoby [2]:

- **Graficky** - v této podobě se nejčastěji setkáme s vývojovými diagramy. Dalšími možnostmi jsou pak různé struktogramy.
- **Slovně** - neboli vyjádřením v přirozeném jazyku. V této podobě jsou nejčastěji vyjádřeny různé návody. Slovně se většinou vyjadřují jednodušší algoritmy, jako jsou již zmíněné návody atd.
- **Matematicky** - pomocí matematických vztahů mezi veličinami, rovnic nebo matic.
- **Programovacím jazykem** - v podobě kódů určitého programovacího jazyka, např.: C++,PASCAL atd.

1.1 Vývojový diagram

Jak již bylo zmíněno výše, každý algoritmus může být vyjádřen několika způsoby. Jedním z grafických vyjádření je vývojový diagram. Vývojový diagram je posloupnost normalizovaných obrazců (bloků) nebo značek. Do těchto bloků jsou následně vepsány symboly, text, popisující operace, které se mají provést. Bloky jsou pak spojeny spojnicemi, pomocí kterých je vždy jednoznačně určen sled operací podle algoritmu. Tyto spojnice mohou být orientované nebo neorientované. Podle polohy jsou spojnice děleny na svislé a vodorovné. V případě kdy je spojnice svislá a neorientovaná postupuje se od shora dolů. U vodorovné neorientované pak zleva doprava. Pokud by se ve vývojovém diagramu postupovalo zprava doleva a zdola nahoru je nutné použít šipky, čímž vzniknou z neorientovaných spojnic orientované.[1], [2]

V předchozím odstavci byly vysvětleny základní obrazce vývojových diagramů. Pro zajištění srozumitelnosti podléhají tyto obrazce určitým normám. V České republice se používala do roku 1996 norma ČSN 36 9030 "Značky vývojových diagramů pro systémy zpracování dat". V roce 1996 vstoupila v platnost nová norma ČSN ISO 5807 "Zpracování

informací. Dokumentační symboly a konvence pro vývojové diagramy toku dat, programu a systému, síťové diagramy programu a diagramy zdrojů systému". Touto normou se připojila České republika k mezinárodní normě ISO 5807:1985 "Information processing - Documentation symbols and conventions for data, program and system flowcharts, program network charts and system resources charts". Tato normalizace přispívá k přístupnosti vývojových diagramů pro širokou škálu uživatelů vzhledem ke sjednocení symboliky a tím i jeho srozumitelnosti. [11]

V oblasti programování se nejčastěji setkáme s následujícími bloky[11], [10]:

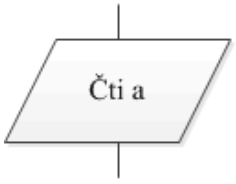
Mezní (koncové) značky

Každý vývojový diagram musí začínat mezní značkou start (viz. obrázek 1) a končit značkou konec (viz. obrázek 1). Označují začátek a konec program popřípadě podprogramů. Každý symbol se může vyskytovat v programu pouze jednou a může mít pouze jeden vstup.

Obrázek 1- Mezní značky [zdroj: vlastní]

Vstup a výstup dat

Tento symbol zajišťuje vstup a výstup dat z nosiče dat nebo vnějšího prostředí do paměti počítače a naopak. Často je tento blok pojmenován čti a piš. Na příkladě (viz. obrázek 2) je ukázka načtení hodnoty „a“ a následně její vypsaní například na obrazovku. Do bloku vstupuje jeden vstup a jeden výstup.

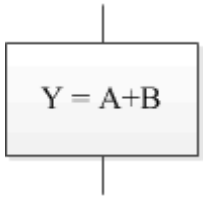


Obrázek 2 - Vstup a výstup dat [zdroj: vlastní]

Zpracování (operace)

V programování je tento symbol využíván pro provedení různé operace, skupiny operací nebo zpracování, který mají za následek transformaci informací. Pod Operací si lze představit přiřazení hodnot, výpočty při nichž dochází ke změně hodnot, atd. V obecné

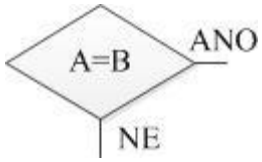
rovině slouží pro jakýkoliv příkaz například: jdi, ulož, atd. Na obrázku 3 je ukázka součtu dvou hodnot „A“ a „B“, který je následně uložen do proměnné „Y“. Do bloku může vstupovat jedna nebo více spojnic ale výstupem je pouze jedna spojnice.



Obrázek 3 - Zpracování [zdroj: vlastní]

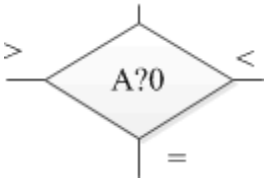
Rozhodovací a přepínací blok

Tento blok plní rozhodovací funkci. Do bloku vstupuje pouze jedna spojnice (vstup) a vystupují dvě spojnice (výstupy). Příslušný výstup je aktivován na základě vyhodnocení podmínky uvnitř symbolu. Zpravidla se jeden výstup označuje „ano“, „+“, nebo „pravda“ v případě, že dojde ke kladnému vyhodnocení. Druhý výstup je pak logicky označen „ne“, „-“, „nepravda“ v případě záporného vyhodnocení. Na obrázku 4 je znázorněn blok porovnávající dvě čísla. Pokud si jsou čísla rovna, bude vyhodnocení kladné a program bude pokračovat po spojnici „ANO“. V opačném případě po spojnici „NE“.



Obrázek 4 - Rozhodovací blok [zdroj: vlastní]

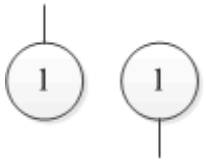
Další variantou rozhodovacího bloku je tzv. přepínač. Oproti rozhodovacímu bloku obsahuje více než dva výstupy. Na obrázku 5 je zjišťováno, zda hodnota „A“ je rovna (=), menší(<), nebo větší (>) nule. Na základě vyhodnocení bude program pokračovat po určité spojnici.



Obrázek 5 - Přepínač [zdroj: vlastní]

Spojka

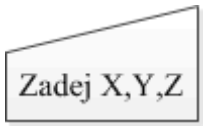
Spojka zprostředkovává přechod z jedné části diagramu do druhé. Užívá se například při velké složitosti programu a velkému počtu větvení pro jeho lepší zpřehlednění. Samotný blok může mít pouze jeden vstup nebo jeden výstup a symbolika v obou spojkách musí být totožná (viz. obrázek 6).



Obrázek 6 - Spojka [zdroj: vlastní]

Vstup z klávesnice

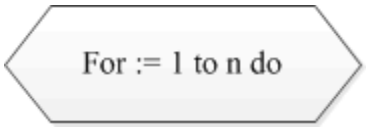
Jak již samotný název napovídá, jedná se o blok pro získání vstupů pomocí klávesnice. Příkladem je zadání čísel nebo textu pomocí klávesnice. Na obrázku 8 je znázorněn příklad zadání hodnot „x“, „y“, „z“ pomocí klávesnice.



Obrázek 7 - Vstup z klávesnice [zdroj: vlastní]

Příkaz cyklu

V případě kdy je znám nebo zadán pevný počet opakování je využíván symbol zobrazený na obrázku (viz. obrázek 8). Při použití cyklu s pevným počtem se bude opakovat určitá operace do dosažení hranice.



Obrázek 8 - Cyklus s pevným počtem opakování [zdroj: vlastní]

2 Tvorba algoritmu

V předchozí kapitole byl vysvětlen pojem algoritmus společně s vývojovým diagramem a jeho základní symboly. Tato část práce se bude zabývat samotnou tvorbou algoritmu.

Při sestavování algoritmu se využívají tři základní struktury [10]:

1. Posloupnost - je řada za sebou navazujících kroků, jejichž pořadí je předem pevně dáno. Každá posloupnost má svůj začátek a konec a žádný její krok nesmí být vynechán. Při algoritmizaci se posloupnost objevuje samostatně nebo je součástí složitějších struktur.

2. Větvení - větvení se používá v případech, kdy mají být podle okolností určité kroky v posloupnosti vynechány, nahrazeny jinými nebo přidány. Větvení obsahuje zpravidla tři části. První část je tvořena otázkou, na kterou existuje kladná nebo záporná odpověď. Druhá a třetí část je krok, který se provede v případě kladné nebo záporné odpovědi. Při neexistenci posledních dvou kroků ztrácí větvení smysl. Při tvorbě algoritmu se mohou objevit tři typy větvení:

- Úplné větvení - jsou zde definovány kroky pro kladnou i zápornou odpověď.
- Neúplné větvení - v tomto případě chybí definice dalších kroků pro kladnou nebo zápornou odpověď. Ačkoliv tento krok chybí, ve vývojovém diagramu se tato skutečnost musí zakreslit.
- Vnořené větvení - znamená existenci kroků pro kladnou nebo zápornou odpověď a tyto kroky jsou tvořeny dalším větvením.

3. cykly - jsou využívány v případech kdy nastane situace, že je nutné zopakování některé činnosti. Počet opakování je závislí na vyhodnocení nějaké podmínky nebo v případě, kdy víme počet opakování, podmínka kontroluje, zda bylo opakování uskutečněno v potřebném počtu. V posledním případě je opakování závislé na vzniku určité situace. Rozeznáváme tři základní typy cyklů. Jedná se o cykly s podmínkou na začátku a s podmínkou na konci. Posledním variantou je cyklus s pevným počtem opakování.

3 Datové typy

V rámci první kapitoly byly vyjmenovány vlastnosti algoritmu a jednou z nich byla hromadnost. Tato vlastnost říká, že program má být využitelný obecně a ne pouze pro jeden případ. Tuto vlastnost si můžeme vysvětlit na jednoduchém případě součtu dvou čísel. Pokud bychom sčítali konkrétní čísla 12 plus 5 a byla by porušena všeobecnost a program by neměl význam.

V rámci programování proto využíváme proměnné, do kterých ukládáme hodnoty. Tato proměnná musí mít dvě vlastnosti, kterými je množina přípustných hodnot a množina operací. Souhrnně se tyto vlastnosti nazývají datový typ. V programování se dělí datové typy na (viz. obrázek 9) :

- Jednoduché
    - ordinální
    - neordinální
- strukturované
- ukazatel
- podprogramy
- ostatní

Podrobnější dělení datových typů je zobrazeno na obrázku (viz. obrázek 10).

<table>
<tr><td rowspan="2">1. Jednoduché datové typy</td><td rowspan="5">Ordinální</td><td rowspan="5">Typ výčet
Typ interval
Typy celých čísel
Typ znak
Typ logických hodnot</td></tr>
<tr></tr>
<tr><td>Neordinální</td><td>Typy racionálních čísel</td></tr>
<tr><td colspan="2" rowspan="5">2. Strukturované datové typy</td><td rowspan="5">Typ množina
Typ pole
Typ soubor
Typ záznam
Typ řetězec</td></tr>
<tr></tr>
<tr></tr>
<tr></tr>
<tr></tr>
<tr><td colspan="3">3.Typ ukazatel
4. Typ podprogram
5. Typ objekt</td></tr>
</table>

Obrázek 9 - Datové typy [zdroj: [2]]

3.1 Jednoduché datové typy

3.1.1 Ordinální datové typy

V tabulce výše je vidět podrobnější dělení jednoduchých datových typů na ordinální a neordinální. Pod pojmem ordinální typy, které pochází z latinského ordo, najdeme hodnoty uspořádané podle velikosti od nejmenšího po největší. U těchto typů je znám počet hodnot a je možné jejich zobrazení na množině celých čísel. V následující části si tyto datové typy přiblížíme.

Výčet

Tento typ je určen seznamem identifikátorů. Z důvodu, že se jedná o ordinální datový typ, je každému identifikátoru přiděleno pořadové číslo. Toto číslo je přiřazeno podle pořadí hodnot ve výčtu, kdy první hodnota má číslo 0 a pro každou následující hodnotu se zvýší pořadové číslo o 1 než u předchozí. Vzhledem k tomu, že se jedná

o obecný datový typ, musí mu uživatel přidělit identifikátor sám. Dále je nutné vytvořit identifikátory pro popis hodnot ve výčtu. [2], [4], [7]

Příkladem výčtového datového typu může být pohlaví. Seznam výčtu bude obsahovat hodnoty „muž“ a „žena“. Hodnotě „muž“ bude přiřazeno číslo 0, neboť se jedná o první hodnotu a „ženě“, která se nachází na druhém místě, bude přiřazena hodnota 1. Datový typ si můžeme označit identifikátorem pohlaví. [2], [4], [7]

Interval

V programování patří interval do obecných datových typů, kterým uživatel musí přidělit vlastní identifikátor a definovat rozsah hodnot patřících do tohoto datového typu. Datový typ interval vždy vychází z jiného ordinálního datového typu, který je nazýván bázový typ. To znamená, že hodnoty na intervalu jsou vždy podmnožinou hodnot určitého bázového typu. [2], [4], [7]

Celá čísla

V různých programech se často pracuje s různými čísly. Tento typ reprezentuje celočíselné datové, mezi které patří nejčastěji používaný typ *integer*. Hodnoty celočíselného typu mohou nabývat hodnot od „-2 147 483 648“ do „2 147 483 648“. Pro čísla, která obsahují desetinnou čárku, se v programování využívá datový typ racionální čísla. [2], [4], [7]

Znak

Char, jak je anglicky označován datový typ znak, slouží k popsání textové informace o velikosti jednoho znaku (například „A“, „B“ atd.). Všechny hodnoty tohoto datového typu jsou obsaženy v tabulce znaků, ve které je každému znaku přiděleno jeho pořadové číslo. V rámci této tabulky jsou hodnoty seřazeny podle těchto čísel a jsou tedy zobrazitelné na množině celých čísel. Znaky lze zapisovat do apostrofů podobně jako řetězce nebo pomocí jeho pořadového čísla tzv. ASCII kódu. V druhém případě předchází tomuto kódu symbol #. [2], [4], [7]

Logické hodnoty

V této práci jsme se již s logickými hodnotami setkali již několikrát například u větvení. Logická hodnota může nabývat hodnot pravda nebo nepravda. Častěji se však

setkáváme s anglickým označením *True* a *False*, kde *True* označuje pravdu a *False* nepravdu. V rámci programování se také můžeme setkat s číselným označením 0 pro nepravdu a 1 pro pravdu. [2], [4], [7]

3.1.2 Neordinální datové typy

Neordinální datové typy jsou opakem ordinálních. Tyto typy pak nejsou zobrazitelné na množině celých čísel. Do tohoto typu patří pouze racionální čísla.

Racionální čísla

V matematice jsou pod pojmem racionální taková čísla, která lze vyjádřit zlomkem. Tyto čísla mají konečný počet desetinných míst. Těmto číslům vzhledem k předem neznámému počtu číslic v desetinné části nelze určit pořadové číslo. Příkladem je otázka kolik čísel je na intervalu <1;2>. Po jedničce následuje 1,1 ale také 1,01 ,1,0001 atd. Těchto čísel je nekonečně mnoho a proto jim tedy není možné přiřadit pořadové číslo. [2]

3.2 Strukturované datové typy

Typ množina

Jedná se o netypický datový typ, ve kterém jsou obsaženy pouze informace, zda množina prvek obsahuje či nikoliv. Počet prvků v množině je omezený a nesmí přesahovat počet 256. [2], [4], [7]

Typ pole

Pole patří mezi základní datový typ, který se vyskytuje ve většině programovacích jazyků. Pole může být jednorozměrné nebo vícerozměrné. Příkladem jednorozměrného pole, které je často označováno jako vektor. Maticí, se kterou se setkáváme například v matematice, jsou v programování označovány dvou- a vícerozměrná pole. Každé pole má pevně daný počet prvků, které musí být stejného datového typu a každý prvek má své jednoznačné označení tzv. index. Index je číslo pole/buňky, ve kterém se hodnota nachází. [2], [4], [7]

Typ soubor

Předchozí datové typy, které byly přiblíženy v předchozím textu, měly společnou vlastnost. Všechny měly uložené své hodnoty v paměti počítače, kde k nim měl program rychlý přístup. Oproti tomu soubor označuje část diskového prostoru obsahující určitá data.

Každý soubor má v rámci diskového prostoru svůj název a přístupovou cestu zajišťující jeho jednoznačnost. [2], [4], [7]

Soubory se mohou dělit z hlediska zpracování podle různých kritérií. První možné dělení je podle použití řídících znaků na soubory textové, netextové s udaným nebo neudaným typem. V dalším dělení jsou rozděleny na soubory určené pouze ke čtení, zápisu nebo určené ke čtení i zápisu. V tomto případě jsou děleny soubory podle druhu prací, které se dají se souborem vykonat. Poslední dělení je závislé na způsobu zpracování dat v souboru na soubory zpracovávané postupně nebo s přímým přístupem. [2], [4], [7]

Typ záznam

Tento strukturovaný datový typ je velice podobný jednorozměrnému poli. U polí musely být všechny prvky stejného datového typu a u záznamu mohou být prvky různých datových typů. V polích se na určitý prvek dotazuje pomocí indexu. V záznamech se k jednotlivým prvkům přistupuje pomocí jejich identifikátorů. Naopak stejnou vlastností u těchto dvou datových typů je pevně daný počet položek záznamu. [2], [4], [7]

Typ řetězec

Během práce jsme se již s podobným datovým typem setkali v podobě datového typu „znak“. Řetězec znaků neboli *string* je posloupnost znaků, která je chápána jako celek. Maximální délka řetězce bývá obvykle 255 znaků. [2], [4], [7]

3.3 Datový typ ukazatel

Jedná se o netypický datový typ, který v sobě nemá uložená konkrétní data, ale pouze informace. Ukazatel je umístěn na konkrétním místě v operační paměti po celou dobu své existence a je používán pro uchování adres prvků v operační paměti počítače. Při deklarování ukazatele její vlastnosti i datové typy zůstávají neměnné a takto nadefinovaná proměnná se bude nazývat statická. [2], [4], [7]

3.4 Datový typ podprogram

V rámci složitějších programů nebo v případech, kdy se vyskytuje určitá operace několikrát, je vhodné použít podprogram. Podprogram lze definovat jako pojmenovanou posloupnost příkazů řešících určitou dílčí úlohu. Mezi výhody podprogramu patří zkrácení zdrojového kódu, který je často dlouhý a složitý. Takto dlouhý program se stává

nesrozumitelným a složitým na odladění chyb a provádění změn. I tyto problémy lze pomocí podprogramu vyřešit. Například, pokud se bude v programu několikrát objevovat příkaz ve stejné podobě, je vhodné vytvořit podprogram. Změny se již nebudou složitě vyhledávat, ale upraví se pouze příslušný podprogram a zamezí se tím možnému přepsání a narušení konzistentnosti celého programu. [2], [4], [7]

V rámci programování se podprogramy dělí na funkce a procedury. Při zavedení funkce je nutné určit datový typ výsledné hodnoty a v těle funkce přiřazovací příkaz, kterým identifikátoru funkce přiřadíme vypočtenou hodnotu. Pro jejich použití se v programování používá pojmů volání podprogramu a v případě procedury příkaz procedury. [2], [4], [7]

3.5 Datový typ objekt

S objekty se běžně setkáme v reálném světě, který je z těchto objektů složen. Každý takový objekt má své vlastnosti. Příkladem objektu může být vozidlo, které je charakterizováno značkou, typem motoru, barvou apod. S podobným principem se setkáváme i v programování. Objektem je zde myšlena množina informací, které vyjadřují programové řešení nebo prvek reálného světa. Objekt je pak definován jako datový typ, který obsahuje datové složky a odkazy na podprogramy, které zajišťují operace nad datovými složkami. [2], [4], [7]

4 Vývoj v oblasti robotiky

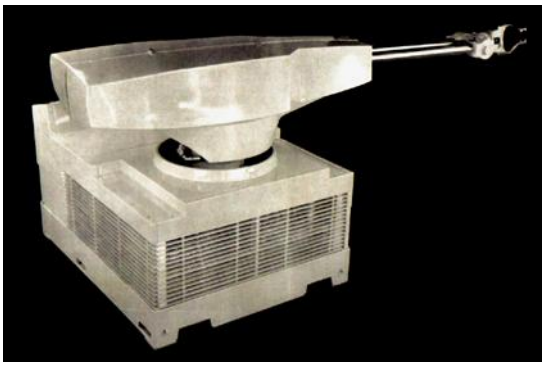
V předchozích kapitolách byla pozornost věnována algoritmizaci a programování. V této části se bude text zabývat vývojem robotiky, jejímž důsledkem je i vývoj softwaru pro jejich programování.

Historie robotiky je spojena s lidským úsilím ulehčit si práci a znásobit svou sílu. V dřívějších dobách často lidé po těžké každodenní práci snili o různých džinech, létajících kobercích a golemech. Tyto bytosti a předměty by pak vykonávali jejich činnost za ně a to sami a automaticky. I v tomto můžeme vidět počátky robotiky, neboť toto snění vedlo k výrobě různých strojů.

Se slovem robot se poprvé setkáváme v díle Karla Čapka R. U. R v roce 1920. V tomto díle vystupují stroje podobné člověku. Zprvu se slovo robot používalo na přilákání pozornosti. První roboti, nebo spíše mechanické stroje, ovšem vznikly již v 18.století. Konkrétně to byl písař od bratří Piera a Henryho Drozů. Tento robot měl podobu člověka a byl schopen psát několik vět. Pokud se vrátíme zpět do 20. století, tak zjistíme, že robotika je spojena s průmyslovou výrobou. S nárůstem poptávky po zboží bylo nutné za pomoci technických prostředků tuto poptávku dokázat uspokojit. Toto bylo patrné především po druhé světové válce, kdy poptávka převyšovala nabídku. Za významné odvětví lze považovat automobilový průmysl, který se dnes bez manipulačních, svařovacích a dalších robotů neobejde. Právě v tomto odvětví byl při výrobě robot poprvé použit. Byl zaveden ve společnosti Ford, kterou vlastnil i Henry Ford. Tento muž zavedl poprvé pasovou výrobu při sestavování jeho slavného automobilu Ford-T. Druhý z mužů, stojících u vzniku této firmy se stal zakladatelem tzv. vědeckého řízení, se mimo jiné zabýval i normováním práce. Toto normování spočívalo v rozkladu složitých činností na jednotlivé úkony pracovníků. Díky této činnosti bylo možné zvýšit efektivnost práce. K tomuto zvyšování pomáhalo přidělení různých nástrojů k ulehčení práce pracovníkům. Ti pak byli schopni vykonávat svůj přidělený úkol rychleji a tím zvyšovat efektivitu výroby. I zde můžeme vidět myšlenku zavádění robotů do výroby. Dalším faktorem ovlivňujícím použití robotů byl technický pokrok, který byl a je předpokladem pro rozvoj výroby nejen robotů, ale i jiných strojů. Do technického pokroku lze zařadit objevy různých materiálů, vynálezy, patenty atd. [8], [9]

Jak již bylo uvedeno výše, právě ve firmě Ford byl využit první průmyslový robot ve výrobě. Jednalo s o robota Unimate 1900 (viz. obrázek 11). Na vývoji se však nepodílela firma, ale byl vynalezen Georgem Devolem a Joseph Engelbergrem. Robot pracoval krok po kroku podle příkazů uložených na magnetickém bubnu. Unimate 1900 byl schopen zvedat břemena o hmotnosti až 226,796 kg. Robot byl využíván nejdříve pro vyndávání horkých vylisovaných součástek vozidel. Později byl inovován a použit i pro jiné činnosti. O dva roky později byl v USA vyvinut robot Versatran (viz. obrázek 10). Robota Versatran bylo také možné vidět v různých variantách. Následně bylo v roce 1964 otevřeno několik institutů zabývajících se robotikou nebo umělou inteligencí jako Massachutess Institut of Technology (M.I.T.) nebo Stanford Research Institute (S.R.I.). Především první zmíněná instituce je známá i dnes. Druhý zmíněný institut se proslavil v roce 1968, kdy vyrobil prvního mobilního robota, kterého pojmenoval Shakey (viz. obrázek 12). Robot byl vybaven televizní kamerou a dalšími senzory, které mu pomáhaly v pohybu. Díky těmto senzorům byl robot schopen samostatně se pohybovat v prostoru. Nevýhodou pro obsluhu byla nutnost ovládat anglický jazyk, ve kterém byl schopen přijímat příkazy. V roce 2004 byl tento robot uveden do "Robot hall of fame", do které byl vybrán i Unimate 1900. [8], [9]

Obrázek 10 - Versatran [zdroj:[3]]



Obrázek 11 - Unimate [zdroj:[13]]

Obrázek 12 -Robot Shakey [zdroj:[3]]

Od 70. let začala převyšovat nabídka nad poptávkou zboží. Bylo to způsobeno tím, že lidé žádali výrobky různého provedení a typu. Následkem tohoto tlaku byla nutnost výroby v menších sériích. Tento vývoj nesl nutnost možnosti přeprogramování techniky a její možnosti přestavby. V této činnosti našli uplatnění průmysloví roboti. [8]

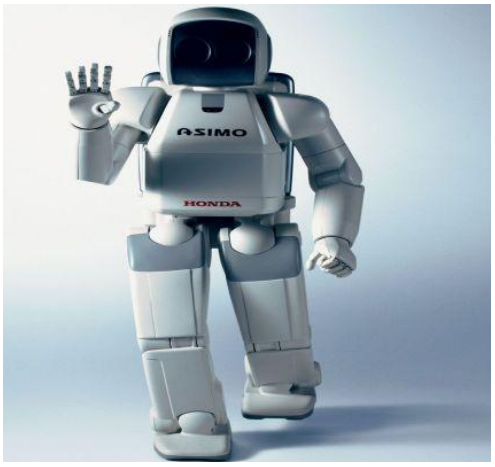
Od 80. let jsou roboti nedílnou součástí průmyslu a jsou masivně nasazováni do výroby. Využívají se pro všechny typy svařování, manipulaci s těžkými břemeny, nanášení barev apod. V roce 1972 bylo používáno na světě 2800 průmyslových robotů a manipulátorů. Ve srovnání s rokem 1984 došlo k nárůstu o více jak 65 000.

Tabulka 1-Vývoj průmyslových robotů [zdroj:[8],[14]]

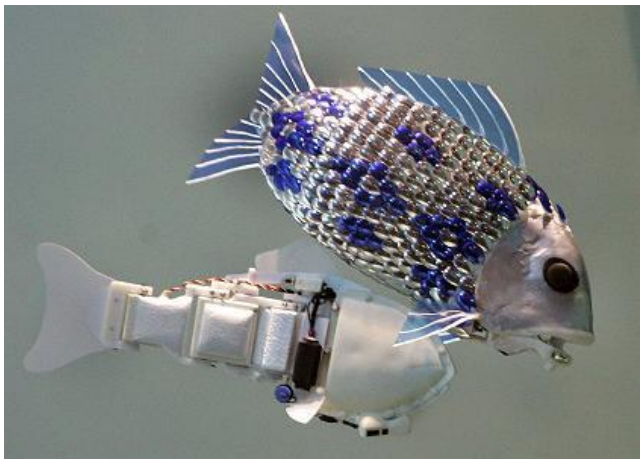
Rok	Počet celkem
1972	2 800
1978	16 000
1980	25 000
1984	68 000
1986	214 000
1990	457 000
1995	700 000
2000	1 300 000
2006	4 500 000
2009	10 930 000

V dalších letech počet robotů narůstal a v roce 2009 se používalo více jak 8 000 000 robotů (viz. tabulka 1). S narůstajícím počtem robotů se také vyvíjela mnoha

směry i robotika. Dnes se setkáváme s roboty v různých odvětvích, jako je lékařství, kosmonautika, vojenský průmysl, ale i v domácnostech. V roce 2000 byl firmou Honda vyroben humanoidní robot „Asimo“ (viz. obrázek 14). Podle tvůrců „Asima“ by měl robot v budoucnosti pomáhat lidem v každodenním životě. Pro českou veřejnost byl „Asimo“ představen v Praze v roce 2004. Další zajímavým projektem bylo vypuštění robotické ryby, která je zobrazena na obrázku 13, vynalezené ve Velké Británii, do španělských vod v přístavu Gijon v roce 2009. Úkolem ryby je shromažďovat informace o znečištění vod, které budou posílány za pomoci technologie *wifi* do počítače, kde se budou zpracovávat. Tím bude umožněno stav vody kontrolovat a reagovat na změny. Tyto uvedené příklady jsou jedny z mnoha, na kterých vědci pracují. [8], [6],[12]



Obrázek 13 - Robot ASIMO [zdroj: [12]]



Obrázek 14 -Robotická ryba [zdroj: [6]]

V této části byl stručně shrnut vývoj robotiky od počátku po dnešek. V budoucnu lze počítat s vývojem robotů, kteří budou podobní člověku. Příkladem tohoto směru může být výše zmíněný robot „Asimo“, který je již dnes schopen napodobovat lidské činnosti jako je chůze, tanec apod.

Díky tomuto vývoji existují dnes programy pro studenty a nadšence v oblasti programování robotů. Jedním z těchto programů je Microsoft Developer studio, kterým se budu zabývat v další části práce.

5 Microsoft Robotics Developer Studio

Jak již samotný název napovídá, jedná se o soubor programů, které jsou využívány především pro programování robotů. Ovšem program může sloužit i k dalším účelům, mezi které patří programování obecně. [5]

Microsoft Robotics Studio (MSRS) je tedy složen z následujících aplikací:

- Visual Simulation Environment
- Command Prompt
- Documentation
- Microsoft Dss Manifest Editor
- Packages
- Run Dss Node
- Visual Programming Language (VPL)

První verze vznikla v roce 2006 za pomocí malé skupiny vývojářů. MSRS byl přímým výsledkem rozhovorů této skupiny s komunitou, která se zabývala robotikou. Výsledkem měla být platforma, která má pomoct vyřešit mnoho překážek, kterým čelí výzkumníci a potencionální zájemci v oblasti robotiky. Skupinu, která vytvořila první verzi programu, tvořilo 11 vývojářů. Program byl založený na *.NET* knihovně, která byla navržena tak, aby usnadnila asynchronní programování. Aplikace nabízí služby orientované na provoz, vizuální nástroje pro tvorbu výukových programů, a dokumentace. Tyto možnosti umožňují jeho využitelnost MSRS jak v komerční tak i nekomerční sféře. [5]

Program prošel několika verzemi a dnes je práce s ním jednodušší. V rámci vývoje se spolupracuje s dodavateli hardware tak, aby je bylo možné integrovat s hardwarem robotů. Dnes je k dispozici verze The Microsoft Robotics Developer Studio 2008 R3, kterou jsem využil pro svou bakalářskou práci.

Až do této verze bylo MSRS nabízeno ve dvou formách. První nabízená forma je určena pro nekomerční účely. Využívána je především studenty, nadšenci a ostatními, kteří mají zájem o robotiku. Druhá forma je určena pro komerční využití. Tato varianta je již

zpoplatněna. Tato licence umožňuje distribuci až 200 kopií všech komponent, které vytvoříte se softwarem. S poslední verzí přišla změna a celý program je bezplatný pro všechny uživatele. [5]

V MSRS můžeme použít některý z podporovaných jazyků *NET Framework*. Lze také využít jazyk *C#, Visual Basic, .NET* nebo skriptovací jazyk Python je-li to jazyk, který programátor preferuje. Program také umožňuje použít grafický programovací jazyk nazvaný Visual ProgrammingLlanguage. [5]

6 Visual Programming Language

Zbytek práce se bude věnovat programu VPL, který je součástí MSRS Program využívá již zmíněný *Visual Programming Language*. Tento jazyk je vhodný pro začínající programátory a studenty. Vizuální rozhraní jim umožňuje používat metodu drag and drop při tvorbě programu. Tento způsob přibližuje software i uživatelům, kteří mají malé nebo nemají žádné zkušenosti s nějakým programovacím jazykem. Mezi nespornou výhodu vyžití tohoto způsobu programování je rychlost s jakou lze program vytvořit. Jazyk je však vhodný i pro pokročilé programátory pro rychlou tvorbu prototypu aplikace. Výsledný program lze konvertovat do aplikací využívající jazyka *C#* . Díky této vlastnosti lze později programy vylepšit a rozšířit ve *Visual studio a .NET framework*. [5]

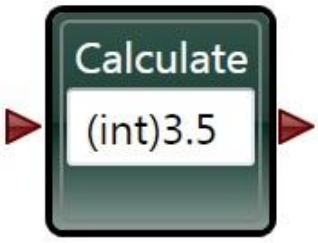
6.1 Datové typy

VPL podporuje datové typy využívané v jazyce *.NET* a *C#*. Mezi tyto typy patří:

Tabulka 2 datové typy [zdroj: vlastní]

Datový typ	hodnoty
Bool	True; False
Int	Célá čísla v intervalu od - 2147483647 do 2147483647
String	Text
Byte	Celá čísla od 0 do 255
Sbyte	Celá čísla od -127 do 128
Char	Znak nebo řada znaků
Decimal	Číslo s přeným zaokrouhlením maximálně na 15 desetinných míst
Double	Reálné číslo zobrazující až 14 desetinných míst
Float	Reálné číslo zobrazující až 7 desetinných míst
Uint	Celá čísla od 0 do 4294967094
Long	-9223372036854775807 do 9223372036854775807
Ulong	Celá čísla od 0 do 18446744073709500000
Short	Celá čísla od -32768 do 32767
Ushort	Celá čísla od 0 do 655355

V tabulce 3 byly uvedeny datové typy využitelné v programu VPL. Nyní bude na příkladě uveden příkaz pro jejich vzájemný převod (viz. obrázek 15). V závorce je uveden datový typ, do jakého má být hodnota převedena, a za závorkou hodnota, která má být převedena. Konkrétně se jedná o převod desetinného čísla do datového typu *integer*. Na příkladě je také patrné, že desetinná čísla se oddělují tečkou.



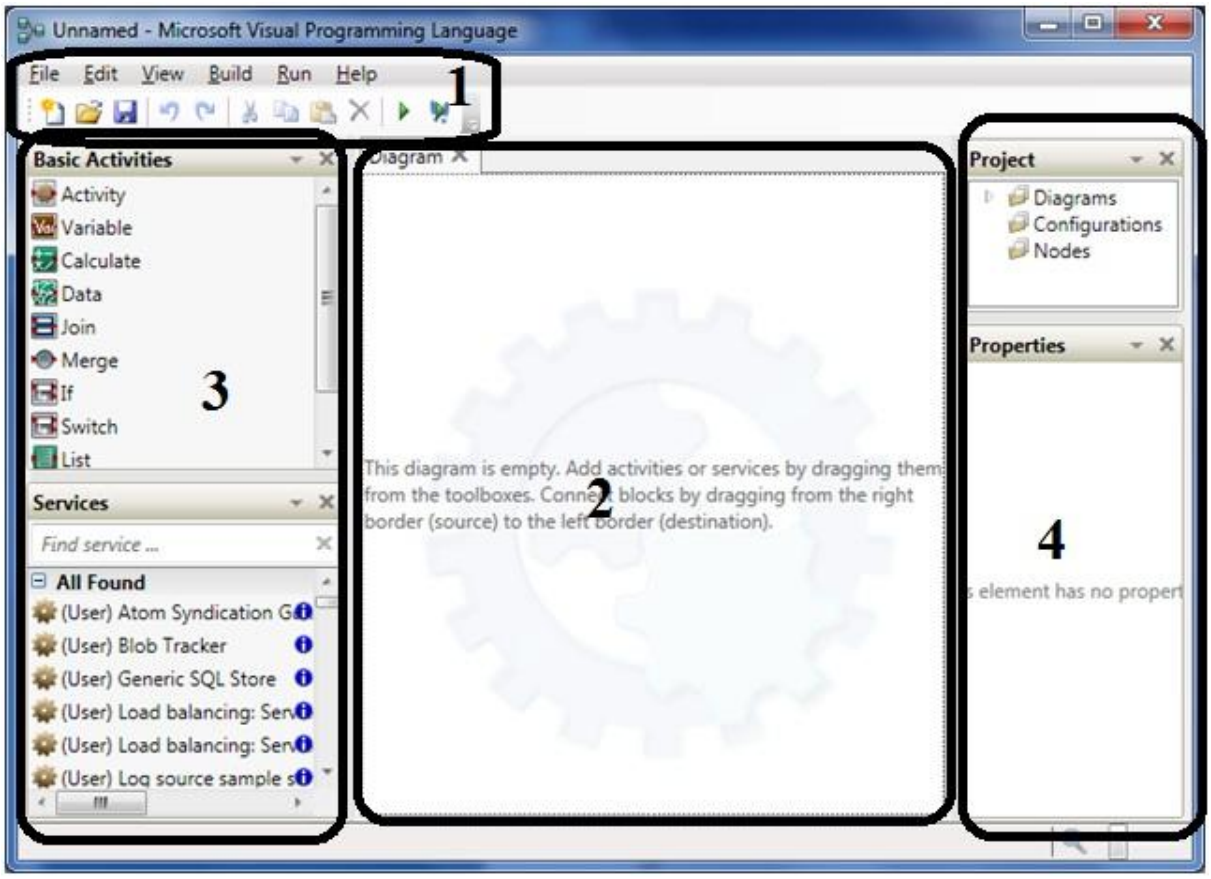
Obrázek 15 -Převod čísla [zdroj: vlastní]

6.2 Základní prvky programu

Samotné aplikace se dá rozdělit na 4 základní části(viz.obrázek 16). V horní části se nachází klasické menu, které obsahuje obvyklé funkce, mezi které například patří otevření, uložení, spouštění programu a také help a informace o aplikaci. Je zde i menu pro spuštění programu nazvané run. V případě, že uživatel chce spustit svůj program, klikne v této nabídce na položku „RUN“.

Druhá část, která je na obrázku označena číslem 2, je plocha, na kterou budou pokládány jednotlivé bloky. Ty jsou poté mezi sebou propojeny spojnicemi. Uživatel má dvě možnosti odkud si potřebný blok přetáhnout, který představuje činnost nebo službu, na tuto plochu. Jedná se o dva druhy aktivit. První jsou tzv. *Basic activity*, které zahrnují bloky pro základní činnosti, vstupní data, výpočty a proměnné. Druhé tzv. *Service*, obsahují servisní činnosti, vestavěné služby, bloky pro určité roboty apod. V těchto aktivitách je možné vyhledávat podle jejich názvu. Tyto nabídky jsou označeny číslem 3.

V pravé části programu se pak nachází panel, na kterém lze zjistit informace o celém projektu nebo jednotlivých blocích. Tento panel je označen číslem 3.



Obrázek 16 - Základní obrazovka VPL [zdroj: vlastní]

6.2.1 Basic activities

V této části si budou postupně vysvětleny jednotlivé činnosti v této nabídce, které jsou následující:

Activities

používá se pro vytvoření nové aktivity. Tato aktivita je definována uživatelem a umožňuje mu si ji vytvořit dle jeho potřeb. Při návrhu vlastní aktivity můžeme použít základní činnosti a služby. Takto vytvořená aktivita může vyvolat oznámení nebo pomocí ní můžeme získat vstupní hodnotu do další aktivity. [5]

Data

Jak sám název napovídá, tento blok slouží jako zásobník hodnot, které se poté využívají v dalších blocích, jako je *Calculate* nebo *Variable*. Hodnoty mohou být datového typu, který podporuje *.NET*. [5]

Variable

V programu slouží pro ukládání hodnot. Tyto hodnoty je možné použít i dále v programu. Stejně tak jako u bloku „data“ mohou být hodnoty datového typu, který podporuje *.NET*. Při deklarování si zde uživatel musí dát pozor na velká a malá písmena, neboť blok je na velikost písmen citlivý. Názvy musí začínat písmenem a obsahovat jen abecední nebo číselné znaky. Žádná interpunkční znaménka nejsou povolena s výjimkou podtržítka.

Calculate

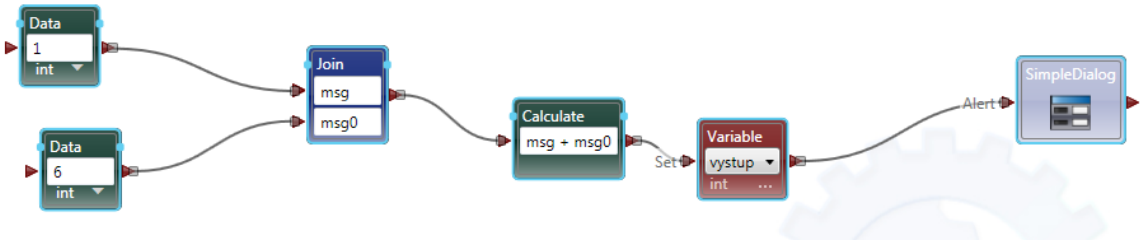
Je blok umožňující s číselnými datovými typy provádět aritmetické funkce, jakými jsou sčítání, odečítání, násobení a dělení. Pro složitější operace se využívá blok *MathFunction,* o kterém se zmíním později. Je možné využít také logické operátory, jako jsou „and“, „or“ nebo „not“. Každé operaci je v programu určen daný symbol, jak je možné vidět v tabulce 3. Pokud je potřeba v tomto bloku zobrazit text je nutné daný text uvést do apostrofů.

Tabulka 3 operace v bloku Calculate [zdroj: [5]]

Aritmetické funkce	
Operace	Symbol ve VPL
Sčítání	+
Odečítání	-
Násobení	*
Celočíselné dělení	/
Zbytek po dělení	%
Logické operace	
And	&&
Or	||
Not	!

Join

Tento blok se používá pro sloučení výsledků z více jak jedné aktivity. Používá se v případech, kdy do další aktivity vstupuje více jak jeden vstup. Například při součtu dvou hodnot, které zadá uživatel, jak je vidět na obrázku 17. Blok využívá logickou spojku „and“.



Obrázek 17 - Součet dvou hodnot [zdroj: vlastní]

Merge

Používá se pro spojení zpráv z dvou a více aktivit. Využití je tedy podobné jako u bloku *Join*. V tomto bloku se využívá logická spojka „or“.

IF

V programu má funkci rozhodovacího bloku. Blok můžeme mít více jak dva výstupy. Tento fakt umožňuje zkrácení programu a lepší přehled. V rámci podmínky je možné použít operátory typu „rovno“, „větší než“, „menší než“ a „různo od“. Symbolika použitá v programu je uvedena v tabulce 4. Výsledkem je hodnota *True* nebo *False*. Stejné operace je možné využít i v bloku *Calculate*. V následující tabulce (viz. tabulka 5) je opět uvedena symbolika pro daný operátor. V případě, že se porovnává text, je nutné ho opět uzavřít do apostrofů.

Tabulka 4 operátory IF bloku [zdroj: [5]]

Operátor	Symbol
Rovno	= nebo ==
Různé od	!= nebo <>
Menší než	<
Větší	>

Switch

Switch má podobné využití jako blok IF. Oproti IF bloku ale nelze provádět operace „menší než“, „větší než“, „různé od“. Blok pouze porovnává, zda je příchozí hodnota rovna hodnotě, která je zadaná v textovém poli Switch. V případě, že tento blok obsahuje text, je nutné jeho uzavření do apostrofů.

List

Pomocí tohoto bloku lze vytvořit seznam hodnot určitého datového typu. Poté může být *List* použit jako vstup do dalších bloků.

List function

Blok se používá pro úpravy prováděné v bloku List. Po úpravách se vytvoří zcela upravený nový List. VPL umožňuje následující úpravy:

- přidání hodnoty na konec seznamu
- spojení dvou listů v jeden nový
- přidaní hodnoty na určitou pozici
- vymazání hodnoty u z určité pozice
- obrácení pořadí prvků v seznamu
- získání indexu určitého prvku v seznamu
- vzestupné seřazení hodnot

 # Comment

Používá se pro popis určité činnosti v programu nebo i celého programu. Jako i v jiných program se při spuštění nezobrazuje. Tento blok pomáhá při orientaci v programu při případných změnách.

6.2.2 Services

V rámci této podkapitoly budou vypsány pouze některé bloky, které byly použity při programování.

Simple dialog

Pomocí této funkce lze snadno vytvořit dialogová okna používaná ve Windows. Takto vytvořený dialog je viditelný po dobu 1 minuty, kdy jej uživatel musí potvrdit. *SimpleDialog* je také limitován omezeným počet znaků, který je schopný zobrazit. Konkrétně je schopný zobrazit maximálně 200 znaků. Využívané jsou především jako jediný způsob vizuální komunikace programu s uživatelem a pro zadání hodnot. VPL nabízí tyto možnosti dialogů (viz. obrázek 18):

Alert dialog- jedná se o blok zobrazující text zadaný uživatelem. Tento dialog obsahuje pouze jedno tlačítko OK, kterým ho lze potvrdit. Využití je především vhodné pro zobrazení výsledků programu.

Promt dialog - V tomto dialogu je možné do textového okna zadat hodnoty, které budou využity v dalších částech programu. Blok ukládá tyto hodnoty jako datový typ *String*. Blok obsahuje také část, kde může uživatel napsat vysvětlivku, jaký typ hodnoty se má zadat.

Confirm dialog - dialog opět zobrazí text, který není možno upravovat. Dialog obsahuje tlačítka OK a Cancel, pod kterými se skrývají pravdivostní hodnoty *True* a *False*.

Obrázek 18 - SimpleDialog [zdroj: vlastní]

Matematické funkce

Blok *MathFunction* poskytuje sadu funkcí, které umožňují provádění běžných matematických operací. Mezi tyto předdefinované operace patří:

- Arcsinus - tato funkce vrátí hodnotu arcsinus v radiánech. Zadaná hodnota musí být v intervalu -1 až 1.
- Arccosinus- funkce vrátí hodnotu arcosinus v radiánech. Zadaná hodnota musí být v intervalu -1 až 1.
- Arctangens- tato funkce vrátí hodnotu arctangens v radiánech. Zadaná hodnota musí být reálné číslo.
- Arctangens2- tato funkce vrátí hodnotu arcTangens v radiánech.
- Cosinus- je opakem funkce arcCosine. Vrací hodnotu cosinus v radiánech ze zadané hodnoty v úhlech.
- Sine - vrací hodnotu sinus v radiánech ze zadané hodnoty v úhlech.

- Tangens - vrací hodnotu tangens v radiánech ze zadané hodnoty v úhlech.
- Sqrt- tato funkce slouží k odmocnění zadané hodnoty, která musí být kladné číslo.
- Todegrees- převádí hodnotu zadanou v radiánech na stupně.
- ToRadians- převádí hodnotu zadanou ve stupních na radiány.
- Random - slouží jako náhodný generátor čísel z intervalu 0 až 1.
- Round- zaokrouhlí vstupní hodnotu na určitý počet desetinných míst, které zadá uživatel.
- Truncate- vrací hodnotu zadaného čísla před desetinou čárkou.
- Exponent- funkce slouží pro umocnění eulerova čísla.
- Logarithm- tato funkce vypočte logaritmus o základu n z čísla x.
- PiandE- tato funkce vrací hodnotu π nebo eulerova čísla.

Textové funkce

Blok *TextFunction* poskytuje sadu funkcí, které umožňují manipulaci s textovými řetězci. Z příslušné nabídky si uživatel opět vybere, jakou operaci chce provést a poté pouze vyplní požadované parametry.

- Contains - funkce zjišťuje, zda se zadané slovo objevuje v textovém řetězci, pokud se slovo v textu nachází, vrací hodnotu *True* a pokud ne, vrací hodnotu *False*. Při použití této funkce je nutné dodržovat velká a malá písmena.
- EndsWith - tato funkce zjišťuje, zda je poslední symbol nebo slovo shodný s tím, který zadá uživatel. Pokud jsou shodné, vrací hodnotu *True*, jinak vrací hodnotu *False*.
- IndexOF - tato funkce vrací index, na kterém se nachází poprvé text nebo jeden a více znaků, zadaný uživatelem. Hledaný symbol může být vyhledán od začátku textu nebo od indexu, který zadá uživatel.
- Insert - pomocí této funkce můžeme do textového řetězce vložit text, znak nebo skupinu znaků. Opět je zde možnost určit, od jakého indexu se má text vložit.

- LastIndexOF - je opakem funkce IndexOf a vrací index, na kterém se nachází naposled text nebo jeden a více znaků, hledaný uživatelem.
- Pad - tato funkce slouží k zarovnávání námi zadaného textu. Text může být zarovnán k levé nebo pravé straně.
- Remove - tato funkce zmaže část textu v zadaném textovém řetězci. Uživatel zde zadá text, pozice, od které se má začít mazat a počet znaku, které se mají smazat.
- Replace - funkce, která nahrazuje v textovém řetězci znak, skupinu znaků nebo slovo.
- StartWith - tato funkce zjišťuje, zda první symbol nebo slovo odpovídá našemu hledanému slovu. Pokud ano vrací hodnotu *True* jinak vrací hodnotu *False*.
- Substring - tato funkce vypíše počet znaků zadaného textu od uživatelem zadaného indexu. V případě, že není zadán počet znaků, které mají být vypsány, vypsaný text se vypíše do konce.
- ToLower - tato funkce převede všechny velká písmena (znaky) na malá písmena (znaky).
- ToUpper - tato funkce převede všechny malá písmena (znaky) na velká písmena (znaky).

7 Programy vytvořené ve VPL

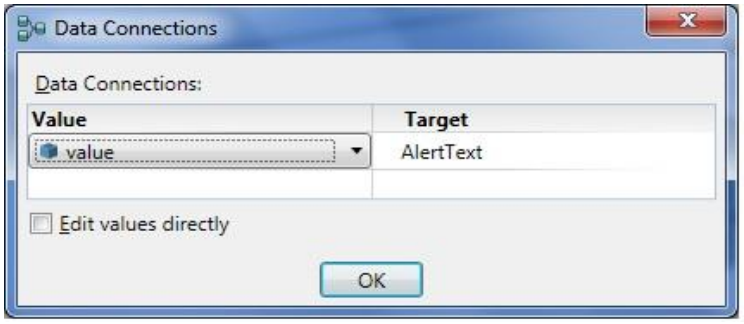
V této kapitole budou popsány některé programy vytvořené ve VPL. Jmenovitě to bude úloha řešící kvadratickou rovnici, řešení přejití po přechodu a vyhledání slova v textu.
V těchto úlohách jsou využity bloky patřící do skupiny *Basic activity* a některé bloky ze skupiny *Services,* které se dají využívat v programování.

V prvním ze tří ukázkových úloh bude řešen problému přejití přes přechod. K tomuto účelu byly použity bloky *SimpleDialog, Merge, Data, Calculate* a blok *IF*. Při řešení kvadratické rovnice bude použit navíc blok *MathFunction* a v poslední úloze ještě blok *TextFunction*.

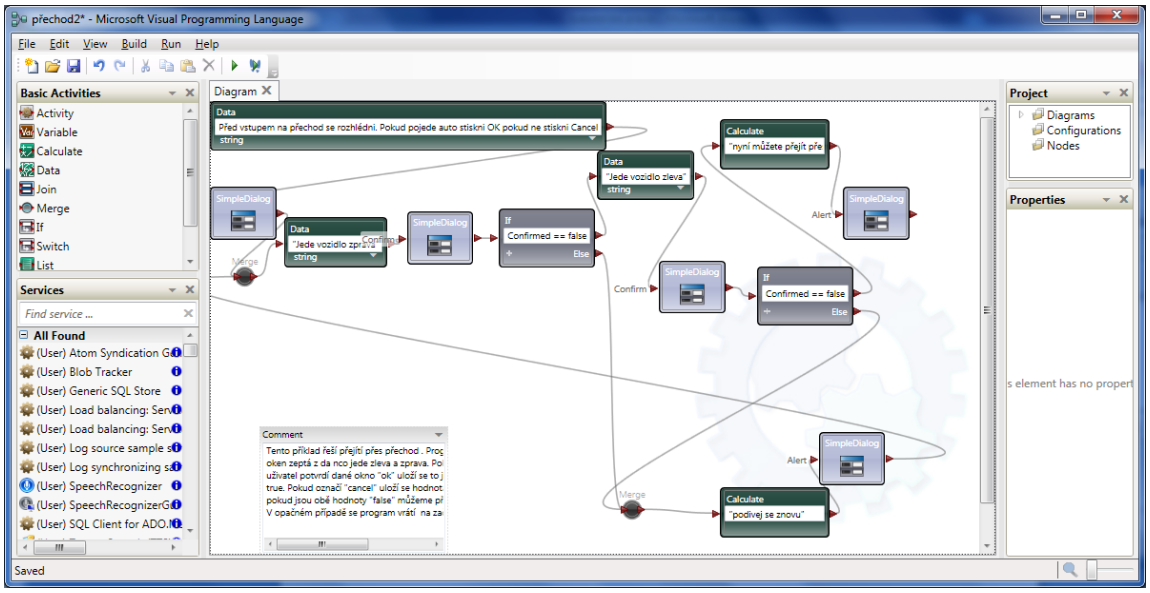
7.1 Přechod

Úloha bude řešit, zda se po silnici nepřibližuje zprava nebo zleva vozidlo a chodec může bezpečně přejít. Celý program je zobrazen na obrázku 20.

Na začátku úlohy bude uživatel informován o jednotlivých funkcích tlačítek pomocí *AlertDialogu*. Při použití tohoto dialogu je nutné změnit parametr *AlertText* na hodnotu *value* (viz. obrázek 19). Poté se ho program, pomocí *ConfirmDialogu*, dotáže, zda jede vozidlo zprava. Pokud uživatel v této fázi potvrdí tlačítko OK, bude to znamenat, že se přibližuje z pravé strany vozidlo a není možné vstoupit na přechod. V případě potvrzení tlačítka Cancel se žádné vozidlo zprava nepřibližuje. Následně se pomocí příkazu *Confirmed* zjistí v *IF* bloku, jaké tlačítko bylo potvrzeno. Pokud bylo potvrzeno tlačítko OK program se vrátí na začátek a opět se dotáže uživatele, zda jede vozidlo zprava. Tento krok je realizován pomocí bloku *Merge*, který spojí větev jdoucí od *IF* bloku a s větví vedoucí od *AlertDialogu*. V opačném případě bude opět dotázán pomocí *ConfirmDialogu,* zda se přibližuje vozidlo zleva. V případě, že bude potvrzené tlačítko OK, objeví se pomocí *AlertDialogu* varovná hláška "nelze přejít podívejte se znovu" a program se vrátí na začátek. V případě potvrzení tlačítka Cancel se objeví varovná hláška "nyní můžete přejít přes přechod".



Obrázek 19 - AlertDialog [zdroj: vlastní]



Obrázek 20 - Přechod [zdroj: vlastní]

7.2 Kvadratická rovnice

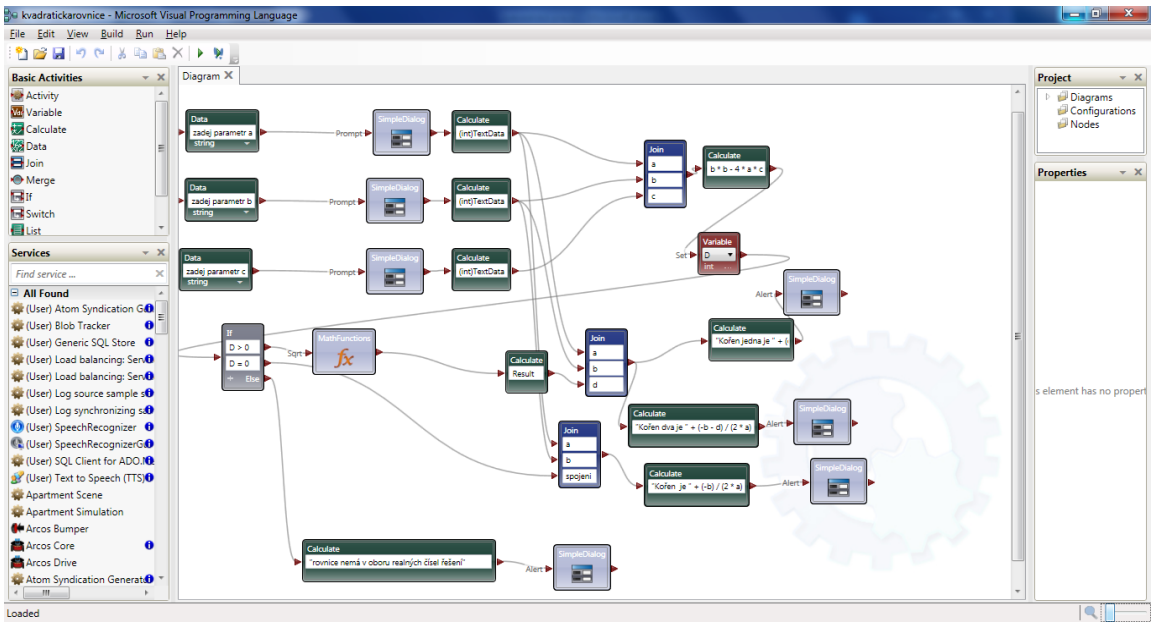
V této úloze bude řešena kvadratická rovnice v oboru reálných čísel. Příklad bude řešen postupně jako klasický výpočet rovnice. Výsledný program je zobrazen na obrázku 21. Podle téhož obrázku je možné sledovat jednotlivé kroky. Nejdříve je nutné zadat jednotlivé hodnoty kvadratické rovnice. Za tímto účelem je použit blok data a *PromptDialog*. Následně vypočítáme pomocí bloku *Calculate* diskriminant. Vzhledem k tomu, že do bloku *Calculate* může vstupovat pouze jeden vstup, je nutné použít blok *Join*. V tomto bloku jsou uloženy jednotlivé příchozí hodnoty z bloků *Calculate*, ve kterých bylo nutné převést vstupní hodnoty na datový typ integer. Převod byl realizován pomocí příkazu *(int)TextData*. *TextData* je hodnota, která je odeslána z bloku

PromptDialog. Poté je vypočítán diskriminant a hodnota je uložena do proměnné D pomocí bloku *Variable*.

V následujícím *IF* bloku bude zjištěno, zda je hodnota proměnné D menší, vetší nebo rovna nule. Pokud je menší, bude program ukončen a na obrazovce bude vypsaná varovná hláška "rovnice není řešitelná v oboru reálných čísel". Tento text je uložen v bloku *Calculate* a zobrazen pomocí bloku *AlertDialog*, jehož parametrem je hodnota z předchozího bloku.

V případě, že hodnota diskriminantu je rovna nule, provede se výpočet jediného kořenu X. Výpočet bude proveden v bloku *Calculate*. Výsledek poté bude následně společně s textem "kořen je" zobrazen v *AlertDialogu*. V bloku *Calculate* je proveden výpočet a pomocí symbolu "+" přidán před výsledek již zmíněný text. Díky této schopnosti není nutné používat další bloky a program je přehlednější.

Posledním možností je výpočet kořenů x_1 a x_2 . Jedná se o případ, kdy je diskriminant, v našem případě hodnota D, větší jak nula. Poté se za pomoci bloku *MathFunction* a funkce *Square Root* vypočítá odmocnina z proměnné D, která je jejím jediným parametrem. Poté za pomoci bloku *Join* spojíme vstupy do bloku *Calculate* a vypočítáme kořeny x_1 a x_2 a zobrazíme výsledky v *AlertDialogu*.

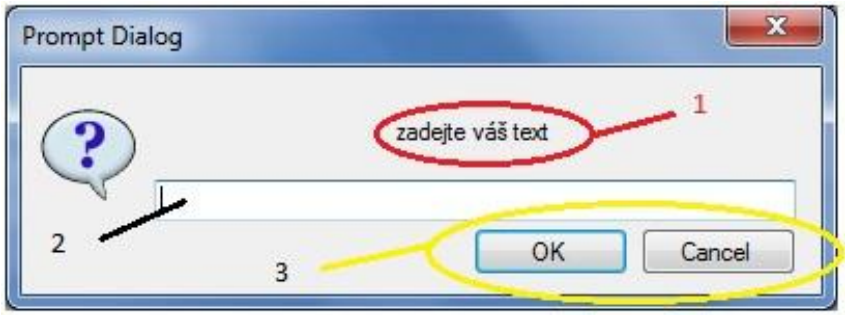


Obrázek 21- Kvadratická rovnice [zdroj: vlastní]

7.3 Hledání slova

Daný příklad řeší, zda se v námi zadaném textu nachází určitý znak nebo řetězec znaků. Výsledný program je zobrazen na obrázku (viz. obrázek 23).

V prvním kroku zadá uživatel text, ve kterém chce hledat určitý znak nebo řetězec znaků. V této části je využitý *PromptDialog* (viz. obrázek 22), ve kterém je možné nastavit dva parametry. Prvním parametrem je tzv. *PromptText*. Tento text se zobrazí v horní části dialogu a nelze ho za běhu programu měnit. Druhý parametrem tohoto bloku je *DefaultValue*. Opět zde může uživatel zadat svůj text, tento text bude viditelný v textovém poli po spuštění bloku a uživatel ho může měnit podle svých potřeb ale pouze po dobu jedné minuty.



Obrázek 22- PromptDialog [zdroj: vlastní]

1 - zobrazení PromptTextu

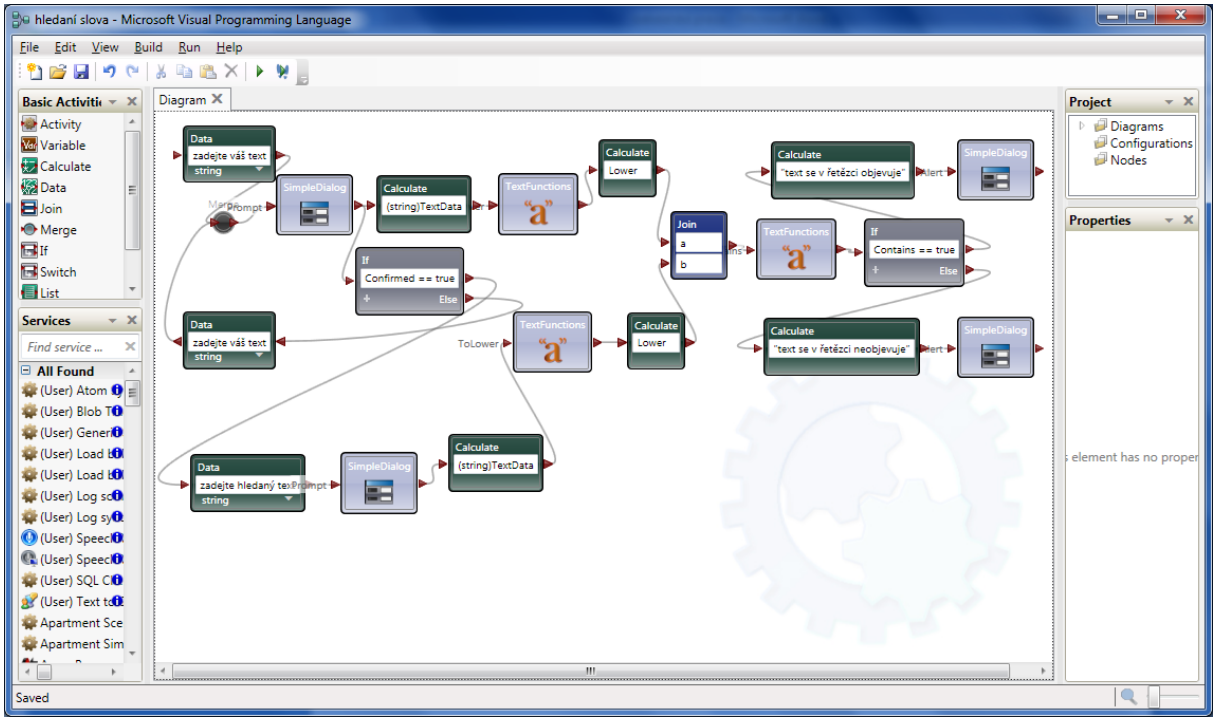
2 - pole pro zadání hodnot

3 - potvrzovací tlačítka

Uživatel po zadání textu potvrdí tlačítko OK nebo Cancel na obrázku 9. Pokud bude potvrzeno OK, uloží se hodnota *True* a v případě potvrzení Cancel pak hodnota *False*. Následně je pomocí příkazu *Confirmed* v *IF* bloku zjištěno, zda příchozí hodnota je *True* nebo *False*. Pokud je podmínka splněna tzn. příchozí hodnota je *True*, je uživatel dotázán, aby zadal hledaný výraz. K tomu je opět využit *PromptDialog*. V opačném případě bude uživatel opět dotázán, aby zadal text. Pro tento účel je použitý blok *Merge*. V této části programu se využívá cyklus s podmínkou na konci.

V předchozích kapitolách bylo zmíněno, že program rozlišuje malá a velká písmena a z tohoto důvodu je nutné oba texty převést na velká nebo malá písmena. K tomuto účelu použijeme blok *TextFunction* a funkci *ToLower* nebo *ToUpper*. V mém případě byla použita funkce *ToLower*, která převede text na malá písmena.

V další části je zjišťováno pomocí bloku *TextFunction* a funkce *Contains*, zda se hledaný výraz v textu nachází či nikoliv. Vstupními parametry funkce jsou hledaný text a textový řetězec, ve kterém se má hledat. V případě, že je hledaný výraz v textu obsažen, vrátí funkce hodnotu *True* a naopak pokud ne vrátí hodnotu *False*. Poté se opět pomocí *IF* bloku zjistí, zda je příchozí hodnota *True*. Pokud ano Zobrazí se na obrazovce pomocí *AlertDialogu* text, že výraz se v řetězci objevuje. V případě hodnoty *False* se naopak zobrazí, že text se v řetězci neobjevuje. Tento text je zapsán v bloku Calculate a poté jeho hodnota vložena do parametru bloku *AlertDialog*. Text by mohl být také zapsán v bloku *Data* nebo uložen do bloku *Variable*.



Obrázek 23 Hledání slova [zdroj: vlastní]

Závěr

Hlavním cílem bakalářské práce bylo vytvořit ukázkové příklady v MSRS. Veškeré programy byly vytvořeny v aplikaci VPL, která je jeho součásti.

V první části byla pozornost věnována procesu návrhu algoritmu, který je označován pojmem algoritmizace. Algoritmus je možné vyjádřit několika způsoby a jedním z nich je i vývojový diagram, o kterém pojednávala další část práce.

Úvodní část druhého většího celku práce popisovala historii robotiky. První využití robotů bylo spjato s průmyslem a firmou Ford, kde byly poprvé využity při výrobě vozidel. V dnešní době se s roboty nesetkáme pouze v průmyslu, ale i v domácnostech, například v podobě chytrých vysavačů. Následující kapitoly jsou věnovány programu VPL a tvorbě programů. V těchto částech byly popsány jednotlivé části aplikace a datové typy využívané programem. Poslední část popisuje některé programy vytvořené v aplikaci VPL.

Výhodou této aplikace je možnost programování bez znalosti určitého programovacího jazyka. V rámci programu jsou spojovány pouze bloky s již přednastavenými funkcemi, které jsou pomocí metody drag and drop umisťovány na pracovní plochu. Tím se práce ve VPL stává rychlou a nenáročnou možností programování.

Při tvorbě jednotlivých programů byly objeveny i některé nedostatky softwaru. Prvním nedostatkem je možnost vytištění textu na obrazovce, který je realizován pomocí bloku *SimpleDialog*. Jeho největší slabinou je 60-ti sekundový interval, po který je viditelný. Pokud není v tomto intervalu dialog potvrzen, zmizí a program poté nemusí podávat správné výsledky. VPL také neumožňuje pracovat s poli. Tato omezující vlastnost neumožnila vytvoření programů, ve kterých se s ním pracuje. Tato skutečnost omezuje varianty programů, které zde lze naprogramovat. Program je však především určen k tvorbě programů pro roboty, který v době vypracování práce nebyl k dispozici a nebylo tak možné vyzkoušet všechny možnosti aplikace. I přes tyto nedostatky však aplikace nabízí kvalitní možnost výuky programování a pro základy je postačující.

Cíle práce byly naplněny a vytvořené programy mohou být využity v předmětu algoritmizace a programování.

Seznam obrázků

Seznam tabulek

Použitá literatura

[1] ČEPELÁK, Jiří. Řešené příklady v jazyku C. Praha: Computer Press 2001. 156s. ISBN 80-7226-575-X

[2] HÁLA, Tomáš. Pascal pro střední školy. Praha: Computer Press, 1999. 279s. ISBN 80-7226-180-0

[3] History of Robotics [online] 2008 [cit.]. Dostupný z WWW: <http://pages.cpsc.ucalgary.ca/~jaeger/visualMedia/robotHistory.html>

[4] MIKULA, Pavel; JUHOVÁ, Kateřina; SOUKENKA, Jiří. Turbo Pascal 7.0 - kompletní průvodce. Praha: Grada 1993. 611s. ISBN 80-7169-010-4

[5] MORGAN, Sara. Programming robotics studio. Washington: Microsoft Press, 2008. 253 s. ISBN 0-7356-2432-1.

[6] Robot Aid. Fish robots search for pollution in the waters. [online] 2001 [cit.]. Dostupný z WWW: <http://www.robaid.com/bionics/fish-robots-search-for-pollution-in-the-waters.htm>

[7] SATRAPA, Pavel. Pascal pro zelenáče. Praha: Neocortex spol.s.r.o 2000. 253s. ISBN 80-86330-03-6

[8] SKAŘUPA, Jiří. Průmyslové roboty a manipulátory [online]. Ostrava: Technické učení, 2007. Dostupné z WWW: <http://www.elearn.vsb.cz/archivcd/FS/PRM/Text/Skripta_PRaM.pdf >

[9] ŠOLC, František; ŽALUD, Luděk. Robotika [online]. Brno: Vysoké učení technické Brno, 2002. Fakulta Elektrotechniky a komunikačních technologií. Dostupné z WWW: < http://matescb.skvorsmalt.cz/robotika_kybernetika/VUT_Brno_Robotika.pdf>

[10] ŠTEFAN, Radim. Úvod do algoritmizace a programování. [online] 2006 [cit.]. Dostupný z WWW:

<http://www.oa-poruba.cz/vp/prg/soubory/uvod_do_programovani.pdf>

[11] TAUFER, Ivan. Vývojové diagramy. [online] 2001 [cit.]. Dostupný z WWW: <http://www.ikvalita.cz/download/kap2.pdf >

[12] TYE, Mike.Bit & Bytes VS Nuts & Bolts [online] 2010 [cit.]. Dostupný z WWW: <http://mikessimpleblog.com/2010/10/bitsbytesvsnutsbolts/>

[13] The Unimate [online] 2009 [cit.]. Dostupný z WWW: <http://pages.cpsc.ucalgary.ca/~jaeger/visualMedia/robotHistory.html>

[14] ZILLIG, Matthias. International federation of robotics[online]. 2009 [cit. 2010-12-14]. Industrial Robot Statistics. dostupné z: < http://www.ifr.org/service-robots/statistics/>