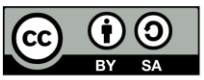
Tento vzdělávací materiál vznikl v rámci projektu
CZ.02.3.68/0.0/0.0/16_036/0005322 **Podpora rozvíjení informatického myšlení**.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY



Digitální technologie v primárním vzdělávání

Martin Dosedla
Zdeněk Hodis
Jiří Hrbáček
Martin Kučera

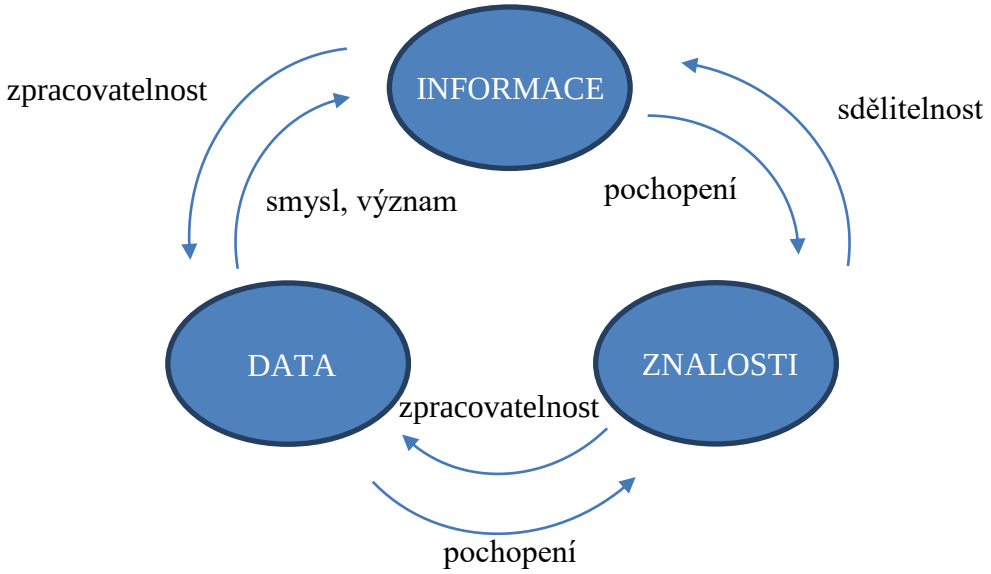
1. Práce s daty různého charakteru (tabulky, grafy aj.), přepis dat z jednoho druhu reprezentace do jiného, vyvozování informací z dat, digitalizace dat.

V následující kapitole se zaměříme na práci s daty různého charakteru (nominální, ordinální, intervalové). Tyto data budeme prezentovat tabelárně, graficky a pomocí kvantitativních ukazatelů. Porozumíme formálním zápisům dat a na základě dat budeme moci vyslovit tvrzení.

1.1. Data

Data můžeme chápat z mnoha pohledů (statistického, informatického aj.). Vždy se však jedná o údaje, které jsou získány pomocí určitého procesu (měření, pozorování aj.). Z těchto dat můžeme následně vyvozovat informace.

V informatice jsou data ukládána typicky ve dvojkové soustavě v nějakém kódu a vytváří tak posloupnost nul a jedniček. Tato posloupnost nám ještě nedává informaci, nicméně je vhodná právě k počítačovému zpracování (ukládání a přenosu). Vztah mezi informacemi, daty a znalostmi můžeme vidět na obrázku 1.1.



Obrázek 1.1 – Informace-data-znalosti.

Statistická data – jsou data, která jsme získali nějakým měřením (fyzikální, pokus, sledování, sportovní, …) nebo jinými výzkumnými metodami (pozorování a záznam, nahrávky, dotazování, rozhovory, …). Data je třeba

ukládat pro jejich další zpracování. Způsob dalšího zpracování a vyhodnocování dat a event. získávání informací z těchto dat záleží do jisté míry na charakteru těchto dat. Rozlišujeme následující typy dat (wikiskripta, 2018):

- **Nominální data** – jedná se o taková data, která nelze přímo vzájemně porovnávat – řadit. Většinou jde o výběr z konečné množiny hodnot. Typickými daty jsou např. pohlaví (Muž, Žena) – nelze určovat pořadí co je více a co je méně. Jedná se tedy o kategorie – kategoriální data. Z dalších příkladů můžeme jmenovat např. rodinný stav, barva vlasů, atd. I přesto, že v zápisu dat můžeme využít číselného kódování (např. M = 1, Ž = 0) nelze tato čísla řadit a srovnávat, stále se jedná o nominální data. Při zpracování takových dat můžeme např. posuzovat četnost výskytu jednotlivých hodnot (tedy kolik je ve skupině můžu a kolik žen) a následně dávat tato data do souvislostí s dalšími získanými daty – vyvozovat závěry / tvrzení.
- **Ordinální data** – stejně jako nominální představují výběr z (konečné) množiny možností s tím rozdílem, že u nich již můžeme určit vzájemné pořadí – tedy určit, která hodnota je „menší“ a která „větší. Příkladem může být nejvyšší dosažené vzdělání nebo třeba různé úrovně znalostí (např. cizího jazyka) – začátečník, mírně pokročilý, středně pokročilý a pokročilý. U ordinálních dat však nemůžeme určovat vzájemnou vzdálenost kategorií. Tedy o kolik je přesně začátečník méně než mírně pokročilý. Ordinální i nominální data představují **kvalitativní** typy dat popisující jisté vlastnosti. Hovoříme také o tzv. **měkkých datech.**
- **Intervalová data** – vyjadřujeme typicky číselně. Můžeme je vzájemně řadit a také určovat o kolik je jedna hodnota větší/menší než jiná. Příkladem může být např. měření teploty ve °C – kdy je např. zřejmé, že 5°C je o 10 menší, než 15°C. Jiným příkladem může být např. měření IQ.
- **Poměrová data** – mají (na rozdíl od intervalových) pevně zadaný nulový bod a můžeme navíc určovat nejen o kolik, ale také kolikrát je jedná hodnota větší než druhá – příkladem jsou SI jednotky – např. měření hmotnosti. Intervalová a poměrová data jsou daty **kvantitativními.**

Kvantitativní data můžeme z jiného hlediska rozdělit na:

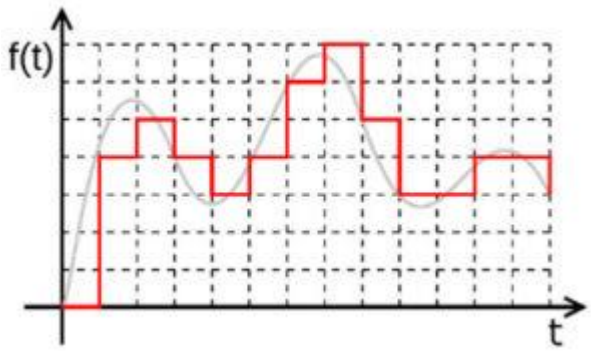
- **Spojité –** nabývají jakékoliv hodnoty – čas – věk respondenta (v praxi však často diskretizujeme – 23 let)
- **Diskrétní** – nabývají pouze předem daných celočíselných hodnost – např. počet automobilů, které projedou po silnici

Při jakémkoli získávání dat a hlavně při zpracování dat je nutné brát v potaz výše uvedené typy dat a volit správné (statistické) metody – např.

u nominálních dat nemá smysl počítat průměr, zatímco u intervalových se jedná popisnou statistickou vlastnost.

Data v informatice: V informatice jsou data ukládána i zpracovávána v binární soustavě v podobě 0 a 1. Všechny výše uvedené typy dat můžeme do této soustavy převést, aby je bylo možné počítačové zpracovávat.

Převodu dat do číselné podoby říkáme **digitalizace – spojité** hodnoty je nutné převést na **nespojité.** Digitalizace analogového záznamu zvuku, videa apod. nebo přímá digitální nahrávka převádí spojitý signál na digitální. Zjednodušeně dochází k vzorkování signálu s určitou frekvencí tak, aby byl zachován jeho průběh (obr. 1.2.). Vzorkovaný signál můžeme již číselně uložit k dalšímu počítačovému zpracování v podobě číselných dat.



Obrázek 1.2 – Informace-data-znalosti *(Otechnice.cz, 2018).*

V oblasti **programování** používáme termín **proměnná** – jedná se o místo, kam ukládáme data v paměti (logicky a abstraktně nikoliv fyzicky). Proměnná může v průběhu programu měnit svoji hodnotu a také mívá často (podle programovacího jazyka) určený typ. Běžné typy, se kterými se můžeme setkat, jsou např.:

- Celé číslo
- Znak
- Reálné číslo (s určitou přesností)
- Řetězec znaků
- Boolean hodnota
- Pole
- Struktura
- aj…

1.2. Práce s daty různého charakteru

Z uživatelského pohledu data (a informace) můžeme zobrazovat a zpracovávat v mnoha podobách zápisů. Nejčastěji se jedná o:

- **Text v čisté podobě nebo formátovaný text**

  Čistý text neobsahuje žádné formátování a jedná se jen čistě o znaky. K jeho zpracování používáme textové editory – např. Poznámkový blok (koncovka .txt). Vzhledem k absenci jakéhokoliv formátování textu je získání informací pouze otázkou čtení, přehlednost a orientace v textu není nijak usnadněna. Přesto se jedná o velmi častý způsob ukládání dat (často pro další zpracovávání). **Formátovaný text** oproti tomu poskytuje mnoho možností jak text zpřehlednit a lépe tak z něj získávat informace. Mezi běžné programy patří MS Word, OO Writer nebo třeba online Google Dokumenty. Základní možnosti formátování textu zahrnují různá písma, velikost, řezy písem, barvy, zarovnávání odstavců, mezery, sloupce, odsazování a mnoho dalšího. Navíc většina programů umí vkládat do textů další prvky dat: tabulky, obrázky aj.

Obrázek 1.3 – Základní možnosti formátování – MS Word.

- **Seznamy – číslované, nečíslované, víceúrovňové**

  Zápis dat v podobě **seznamů** je vhodnější pro stručný a přehledný zápis menšího počtu znaků. U seznamů je výhodou přehlednost, možnost organizace položek seznamu – hlavně řazení podle abecedy nebo podle jiné vlastnosti položek. Obecně rozlišujeme seznamy číslované a nečíslované (číslování může být však řešení i jinak než číslicemi, např. písmeny). Podstatné je, že nečíslované seznamy využíváme tam, kde není podstatné pořadí jednotlivých položek seznamu a naopak. Pro nečíslované seznamy můžeme využít i různé grafické odrážky.

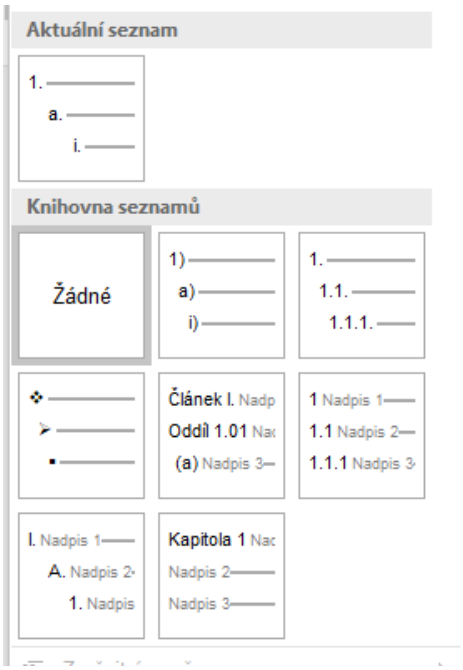
  Příkladem vhodného použití nečíslovaného seznamu může být třeba nákupní seznam, obsah lékárničky, apod. Příkladem číslovaného seznamu např. pracovní postup nějaké činnosti.

  Víceúrovňové seznamy pak umožňují zápis složitějších vztahů mezi položkami tím, že jednotlivé prvky seznamů vnořují do nadřazených:

  I. Fyzika

a. Mechanika
    i. Páka
    ii. Kladka
b. Optika

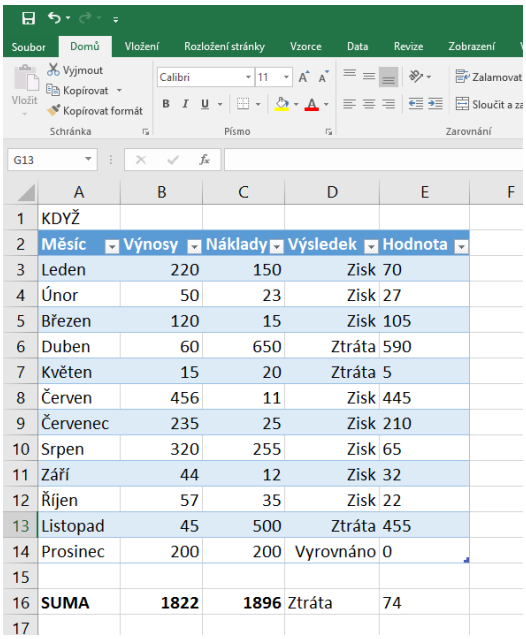
II. Chemie
a. Anorganická
b. Organická



Obrázek 1.4 – Vybrané možnosti práce se seznamy – MS Word.

- **Tabulky**

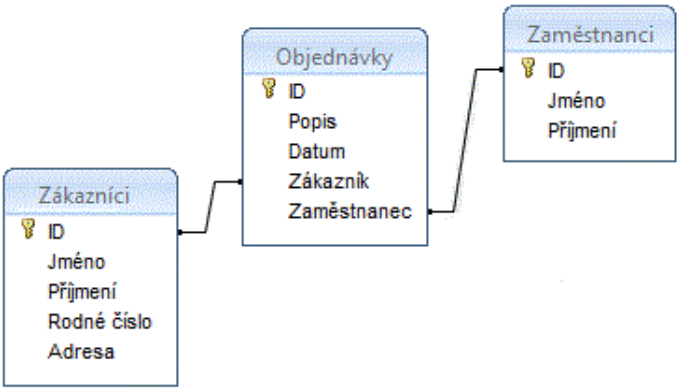
  Organizace dat do řádků a sloupců a jejich kartézský součin tvoří dvourozměrnou datovou strukturu pro zápis dat – tabulku. **Tabulky** se široce využívají pro záznam nejen měřených údajů. Tabulkou může být např. rozvrh hodin, výsledky měření ve fyzice, sportovní výsledky aj. Ke zpracování tabulek slouží primárně tabulkové procesory – např. MS Excel. Ty umožňují nejen tvorbu samotných tabulek, ale hlavně provádět v tabulkách výpočtu pomocí vzorců a funkcí, data v tabulkách řadit a třídit a také z tabulek tvořit grafy viz dále. Pro jednoduché tabulky však je možné použít i textové procesory jako je MS Word.



	A	B	C	D	E	F
1	KDYŽ					
2	Měsíc	Výnosy	Náklady	Výsledek	Hodnota	
3	Leden	220	150	Zisk	70	
4	Únor	50	23	Zisk	27	
5	Březen	120	15	Zisk	105	
6	Duben	60	650	Ztráta	590	
7	Květen	15	20	Ztráta	5	
8	Červen	456	11	Zisk	445	
9	Červenec	235	25	Zisk	210	
10	Srpen	320	255	Zisk	65	
11	Září	44	12	Zisk	32	
12	Říjen	57	35	Zisk	22	
13	Listopad	45	500	Ztráta	455	
14	Prosinec	200	200	Vyrovnáno	0	
15						
16	**SUMA**	**1822**	**1896**	Ztráta	74	
17						

Obrázek 1.5 – Tabulka Nákladů a Výnosů - příklad – MS Excel.

Hlavními výhodami je viditelný vztah mezi řádky a sloupci. Tabulky jsou také základní prvkem, ze kterých jsou tvořeny databáze (relační databáze). Databází je spojení více tabulek, mezi kterými existují vztahy. Např. tabulka žáků a jejich údajů, tabulky učitelů a jejich údajů a následně vztah: učitel učí žáka. Jiný příklad viz obrázek níže.

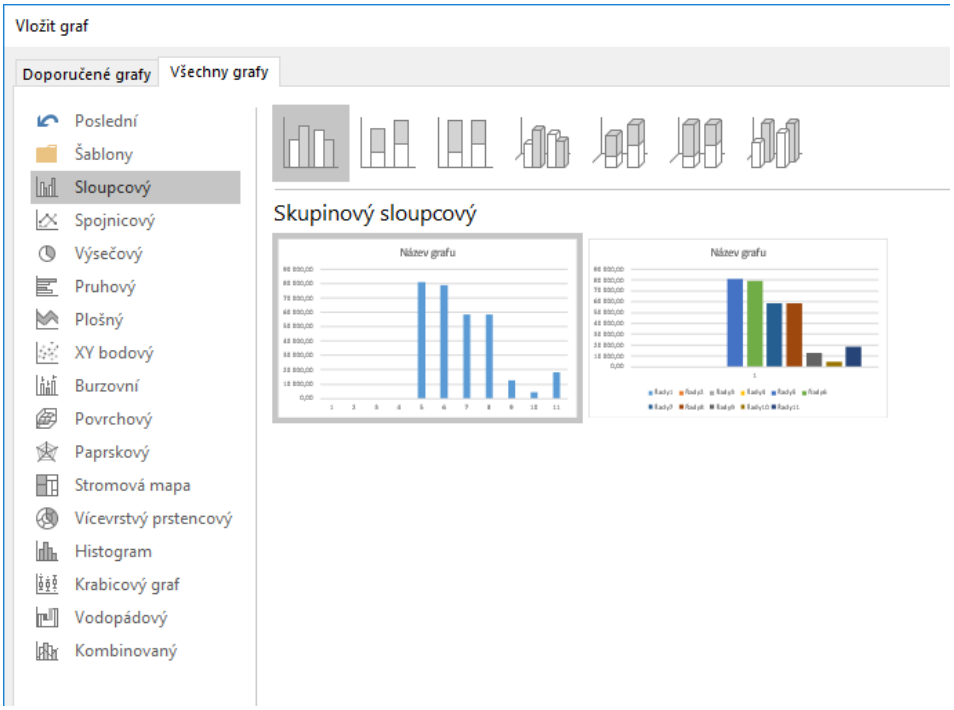


Obrázek 1.6 –Příklady vztahů mezi tabulkami v databázi (tabulky zjednodušeny na záhlaví) *(http://www.databaze.chytrak.cz/modely.htm).*

- **Grafy**

  Graf zobrazuje graficky data v přehlednější podobě, ze které mohou být na první pohled zřetelné skutečnosti, které v číselné nebo tabulkové podobě nejsou tak přehledné. **Grafy** většinou konstruujeme na základě tabulkového zápisu. Tedy pro tvorbu grafů můžeme využít tabulkový procesor nebo podporu pro tabulky a grafy v textovém procesoru.

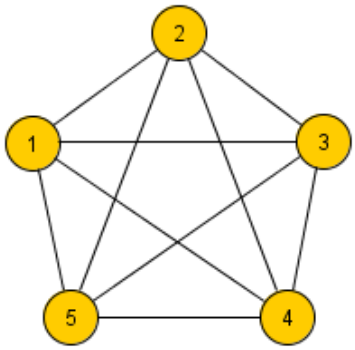
Tabulkové procesory nabízí celou řadu typů grafů. Vždy je třeba zvolit takový typ grafu, který nejlépe a nejpřehledněji zobrazí požadovaná data (např. nevolit spojitý graf pro nespojitá data).



Obrázek 1.7 – Přehled typů grafů v MS Excel.

U grafů je velmi důležité, aby v nich data byla jasná, tedy bylo zřejmé, co zobrazují, jaké mají hodnoty. Zajistíme to vhodným popisem os a hodnot na osách, názvem grafu, legendou apod. Jen správně popsaný graf může přehledně zobrazit data.

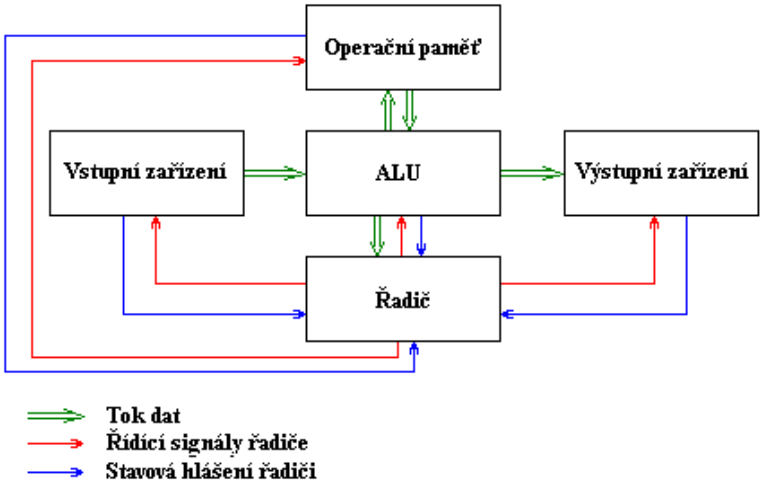
Kromě výše uvedených typů grafů v tabulkových procesorech máme ještě jiné typy grafů. V matematice a informatice grafem rozumíme dvojici G=<V,E>. Zjednodušeně se jedná o množinu vrcholů (uzlů) grafu V a množinu hran E, které vrcholy propojují. Takovým grafem je tedy i graf na obr. 1.8.



Obrázek 1.8 – Graf *(https://www.algoritmy.net/article/1369/Graf).*

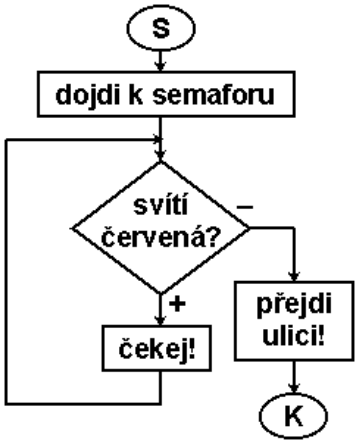
- **Schémata a diagramy**

  Schémata a diagramy jsou do jisté míry podobné grafům. Existuje mnoho druhů schémat a diagramů – elektrického/elektronického obvodu, schéma linek městské dopravy, schéma rozvodů tepla aj.



Obrázek 1.9 – Von Neumannovo schéma činnosti počítače
(https://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/VNEUM.HTML).

Velmi časté je využití **diagramů** nějaké činnosti – např. vývojový diagram při algoritmizaci či programování.



Obrázek 1.10 – Vývojový diagram
(https://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/VNEUM.HTML).

- **Obrázky a symboly, značky**

  Technicky vzato jsou **obrázky** i výše uvedené diagramy, grafy aj. Jedná se o rastrový grafický formát (jpeg, gif, png, …), případně o vektorovou grafiku. Z pohledu informací a dat je však podstatné, co je na obrázku uvedeno. Obrázek ilustruje skutečnost, velmi rychle dává představu, o dané skutečnosti. Není nutný dlouhý slovní popis, ale stačí fotografie např.

automobilu a hned můžeme získat celou řadu informací (barva, typ, značka, prostředí apod.). Předávání informací v podobě obrázků nedává tolik prostoru k představivosti.

Pod **symboly** rozumíme jednoduché (grafické) **značky** – často vektorově zpracované. Symbolem může být i písmeno - symbol pro „A“ je A. Symboly a značka se využívají i pro konstrukci schémat a diagramů (např. značka pro žárovku v elektrickém obvodu). Symboly nalezneme na dálkovém ovladači, na pračce, pro orientaci v obchodech, nádražích a v mnoha dalších případech.

Obrázek 1.11 – Symbol pro praní na 60° a značka/symbol pro parkoviště pro invalidy *(https://www.exesport.net/praci-symboly/ a http://www.dopravniznaceni.com/IP-12-O1-vyhrazene-parkoviste-pro-invalidy-d507.htm).*

Pro zpracování schémat, diagramů, značek, obrázků, apod. na počítači využíváme grafické programy. V omezené míře můžeme značky zpracovávat např. v textových editorech – pomocí nástrojů kreslení. Pro jednoduché značky jsou vhodnější vektorové grafické editory a pro fotografie rastrové editory.

- **Zvuk**

  Zvuk ať už v podobě mluveného slova, hudby nebo jiných zvukových projevů a nahrávek (houkačka, štěkot psa, bouřka, …), na rozdíl od všech výše uvedených druhů dat, vnímáme sluchem a nikoli zrakem. Digitální záznam zvuku můžeme zpracovávat v příslušných programech (např. Audacity) a dále s ním pracovat.

- **Video a animace**

  Video a případně animace jsou ve svém principu rozpohybované obrázky (obsahující i zvukovou stopu). **Animace** je velmi vhodné využít v kombinaci s některými výše uvedenými druhy dat. Animace schématu jaderné elektrárny s principem činnosti může být o mnoho názornější než pouze statické schéma doplněné popisem činnosti v podobě textu. Videozáznam reálného světa zaznamenaný na kameru, můžeme dále

v počítači upravovat a zpracovávat, např. v programech určených pro střih videa (ShotCut).

1.3. Závěr

Všechny typy dat můžeme zpracovávat na počítači v jednoduché podobě vhodné i pro žáky prvního stupně základních škol. Úkolem dobrého učitele je, aby navrhnul pro žáky konkrétní projekty a cvičení na zpracovávání dat. Může se jednat o tvorbu vizitek, plakátů, třídních časopisů a novin, návrhy diagramů nebo kresbu jednoduchých schémat, doplňovaček, křížovek, úpravu obrázků a mnoho dalšího.

Literatura

OPLETALOVÁ, M. *Analýza dat v ekonomii.*[online] VŠEM. Dostupné z https://docplayer.cz/1213610-Analyza-dat-v-ekonomii.html

Otechnice.cz. *Digitalizace, kodeky aneb jak to vše funguje.*2017[online] Dostupné z https://www.otechnice.cz/wp-content/uploads/2017/07/Obra%CC%81zek2-300x181.jpg

Wikiskripta. *Typy dat.*2018 [online]. Dostupné z https://www.wikiskripta.eu/w/Typy_dat

Databáze. *Databázové modely.*[online]. Dostupné z http://www.databaze.chytrak.cz/modely.htm

Algoritmy.net *Graf.* [online]. Dostupné z https://www.algoritmy.net/article/1369/Graf

FI MU. *Von Neumannovo schéma.* [online]. Dostupné z https://www.fi.muni.cz/usr/pelikan/ARCHIT/TEXTY/VNEUM.HTML

Stránky k výuce informatiky. *5. Vývojové diagramy k slovně zapsaným algoritmům.*[online] Dostupné z http://www.ivt.mzf.cz/algoritmizace-a-programovani/uvod-do-algoritmu/5-vyvojove-diagramy-k-slovne-zapsanym-algoritmum/

Harex dopravní značení. *IP 12+01.*[online]. Dostupné z http://www.dopravniznaceni.com/IP-12-01-vyhrazene-parkoviste-pro-invalidy-d507.htm

EXESPORT.net. *Prací symboly.* [online]. Dostupné z https://www.exesport.net/praci-symboly/

2. Základy algoritmizace a programování.

V následující kapitole se zaměříme na základy algoritmizace a programování. Porozumíme sestavování jednotlivých kroků symbolického zápisu algoritmů, čtení programů, jejich testování a odstraňování chyb, orientace v blokově orientovaném dětském programovacím jazyce. Zaměříme se na řazení příkazů, práce s programovacími strukturami: cykly, podmínky, podprogramy. Práce s objekty, spouštění programů událostmi, komunikace mezi objekty.

2.1. Algoritmizace

Programování a algoritmizace je zadávání po sobě jdoucích příkazů v počítači, postupně řazených kroků tak, aby se dosáhlo kýženého výsledku. Tyto příkazy musí být dostatečně jednoduché, aby jim počítač porozuměl a dostatečně jasné a přesné, aby nedocházelo k možným záměnám. Pakliže kroky jasně a přesně formulované nejsou, počítač dělá chyby, naše programování a formulace příkazů nebyla dostatečně přesná.

Pochopení souvislostí a správné posloupnosti je podmíněno dosažením přiměřené úrovně v předchozích zmíněných oblastech. Posloupností se rozumí řada pokynů, které následují za sebou (jsou v určitém pořadí) (Maněnová, Pekárková 2018).

Algoritmus je přesný postup, jakým je možné daný úkol vyřešit. Jedná se o určité kroky, které vedou k řešení. Čím méně kroků tvoří algoritmus, tím je kvalitnější. Nezáleží tedy pouze na správném sledu kroků, ale i na jejich počtu. Robotická hračka se dostane k cíli pomocí různých variant kroků (kombinací základních pohybů), ale jen jedna cesta bude nejkratší.

Algoritmizaci lze rozdělit do několika etap (Bromová, 2012):

- Formulace problému.
- Analýza úlohy.
- Vytvoření algoritmu.
- Sestavení programu.
- Odladění programu.

Formulace problému spočívá v přesném vymezení požadavků, určení výchozích hodnot a požadovaných výsledků. Výsledky je nutné dále specifikovat a říci, jakou mají mít formu.

Při analýze úlohy se ověřuje řešitelnost problému a provádí se hrubý nástin řešení.

Vytvoření algoritmu úlohy je realizováno sestavením přesného sledu operací vedoucích k vyřešení dané úlohy. Tento algoritmus pouze udává postup řešení daného problému. Samotný problém neřeší.

Následuje sestavení programu dle daného algoritmu. Tento program je již napsán v konkrétním programovacím jazyce.

Posední fází je odladění programu, během něhož se odstraní z programu chyby. Závažné jsou chyby ve funkčnosti, kdy je nutné se vrátit zpět k analýze úlohy a sestavení algoritmu a nalezení chyby na této úrovni.

V rámci činností rozvíjejících algoritmické myšlení bychom chtěli, aby dítě (Maněnová, Pekárková 2018):

- Dokázalo seřadit obrázky ve správném pořadí.
- Vyprávělo příběh na základě obrázků.
- Dokázalo zdůvodnit pořadí.
- Popsalo obrázky jako sled pokynů (v případě, že k tomu bude vhodný úkol).

Základní pravidla k podpoře algoritmického myšlení:

Tento postup se doporučuje využívat u všech aktivit proto, aby dítě využilo svoji úroveň reflexe postupů, uměl pustit chybný plán řešení, dovedlo najít jinou strategii a tu vyzkoušet (Maněnová, Pekárková 2018).

- Dítě si algoritmus vyzkouší – zahraje si hru.
- Reflektuje svůj výsledek – popisuje a vypráví.
- Analyzuje problém – najde chybu, pokud tam je.
- Má nápad – ví, jak chce řešit jinak (nová idea).
- Přeformuluje algoritmus – opraví podle nově nalezené souvislosti.

Obrázek 2.1 – Ukázka řazení příkazů pomocí kartiček (Arrow Jumping Game, 2015).

Algoritmizace se využívá ve speciálních výukových programech. Jsou to např.:

- Karel,
- Imagine Logo,
- Baltík/Baltazar,
- Petr,
- Scratch.

Většina programů je vhodná pro starší děti, jsou však i výjimky – např. Baltík / Baltazar. Baltík je výukový multimediální programovací a kreslící nástroj pro děti a mládež. V současné době již existují čtyři verze tohoto populárního dětského programovacího nástroje. Prvním, v této řadě, byl SGP Baltazar. Následoval SGP Baltík 2, vhodný pro děti ve věku 4 – 8 let. Poté přišel SGP Baltík 3, který je určen dětem od 6. let. Zatím posledním je SGP Baltie 4 C#, který již podporuje objektově orientované programování a je trojrozměrný. Z tohoto důvodu je také vhodný až pro starší děti a to od 10. let věku (Bromová, 2012).

Obrázek 2.2 – SGP Baltík3 (https://www.instaluj.cz/baltik).

Hračka je v obecném významu předmět, který podporuje základní dětskou potřebu či činnost – hru. Specifickou skupinou hraček jsou robotické hračky. Jednou z nejrozšířenějších a nejvhodnějších hraček je Bee-bot. Robotická hračka Bee-bot představuje velmi jednoduchého robota, kterého je nutné naprogramovat.

2.2. Řazení příkazů, práce se strukturami a objekty

Z uživatelského pohledu můžeme data (a informace) zobrazovat a zpracovávat v mnoha podobách zápisů. Nejčastěji se jedná o:

Programování, což je výborný nástroj pro to, naučit se vytvářet algoritmy, tedy postupy činností, které někdo či něco automaticky vykoná a tím dochází k rozvoji informatického myšlení. Vykonavatelem takovýchto činností může být např. programovatelná robotická **včelka Bee-bot.**

Obrázek 2.3 –programovatelná robotická včelka Bee-bot.

Programování probíhá formou hry – dítě/žák má nejčastěji vymyslet algoritmus, který včelku „doveze“ z počátečního místa do cílového místa. Včelka jezdí po čtvercové síti s rastrem 15 cm. Ke zvýšení motivace lze použít nejrůznější tematicky zaměřené podložky nebo vkládat vlastní obrázky pod průhlednou folii.

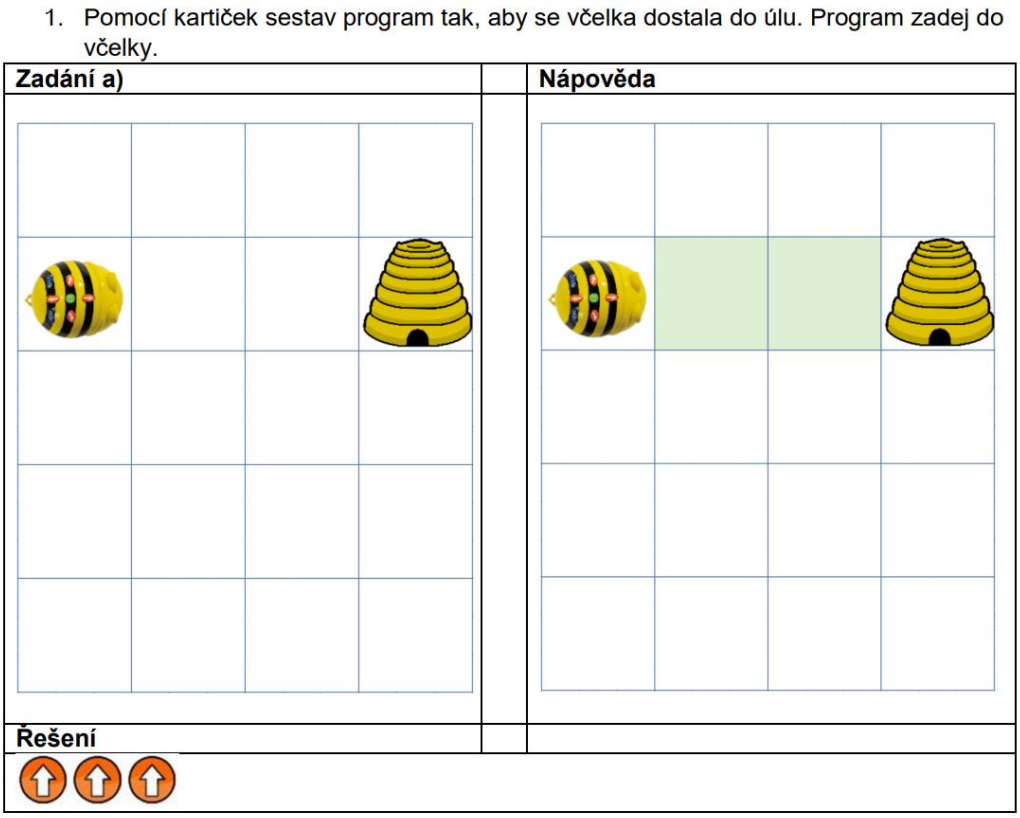
Obrázek 2.4 –příklad úlohy s využitím tematicky zaměřené podložky.

Program se tvoří jednoduše – stisknutím tlačítek na hřbetě včelky (pohyb dopředu, dozadu a otočení vlevo/vpravo). Každý stisk je přitom automaticky uložen do paměti robota. Stiskem tlačítka „GO“ se celá sekvence (až čtyřiceti příkazů) začne vykonávat. Žák tedy nejprve sestaví program, aniž vidí, jak se program postupně vykonává, jak „to dopadne“. Musí si výsledek nejprve představit. Teprve poté, co je včelka naprogramovaná, ji žák spustí a kontroluje, co vykonává. Když žák sdělí své příkazy včelce špatně, např. ve špatném pořadí, včelka vykoná něco jiného, než žák chtěl. Žáci si osvojují, že tvorba algoritmu představuje sled příkazů, které vedou k cíli, k vyřešení určitého problému.

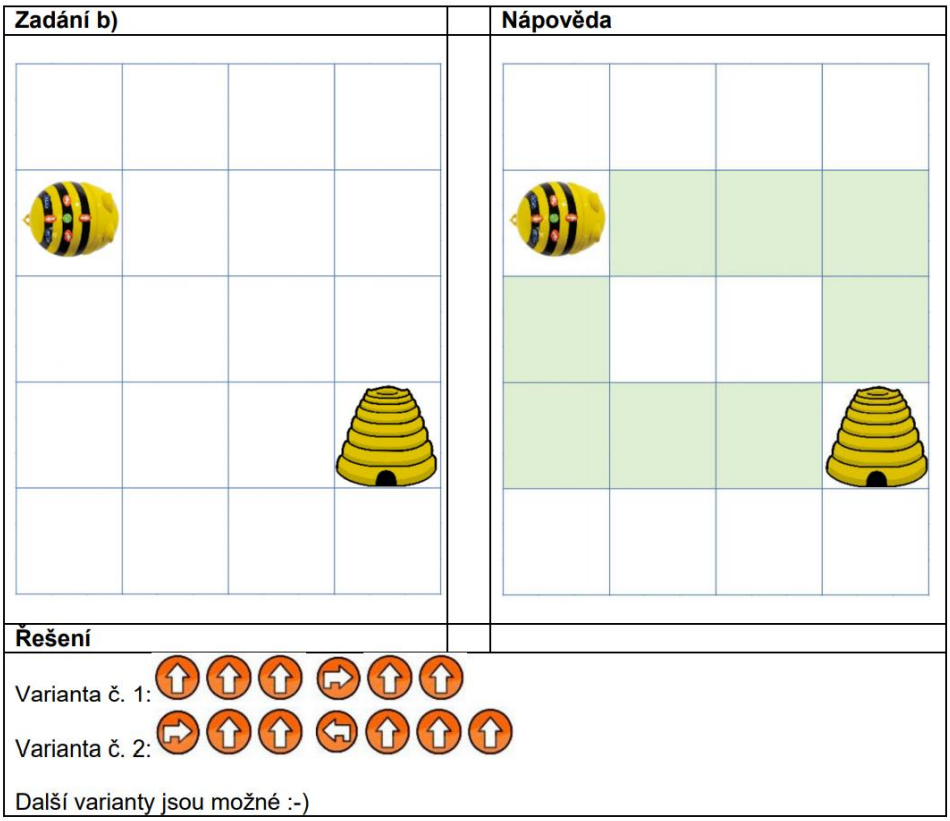
Pomocí robotické hračky Bee-bot můžeme rozvíjet algoritmické kompetence:

- ověření, že program pracuje správně;
- navrhování řešení (vybrat vhodnou cestu k cíli);
- určení cílového místa, kam daný program včelku doveze;
- určení počátečního místa, odkud včelka vyjede, aby při daném programu dojela do daného místa;
- hledání chyby v programu (při jeho vykonávání);
- testování programu (najít způsob, jak ověřit, že program pracuje správně);

- ladění programu (zjednodušení programu nebo jeho úprava, aby správně reagoval v různých situacích);
- zapsání programu (např. pomocí šipek na papír);
- přečtení programu a jeho vložení do robota;
- hledání chyby v napsaném programu (šipky na papíře);
- optimalizace (úvahy o nejkratším programu nebo o nejkratší cestě na dané místo);
- opakování, úvahy o řetězení programů (co se stane, když se program vykoná dvakrát po sobě) (Picka, 2018).

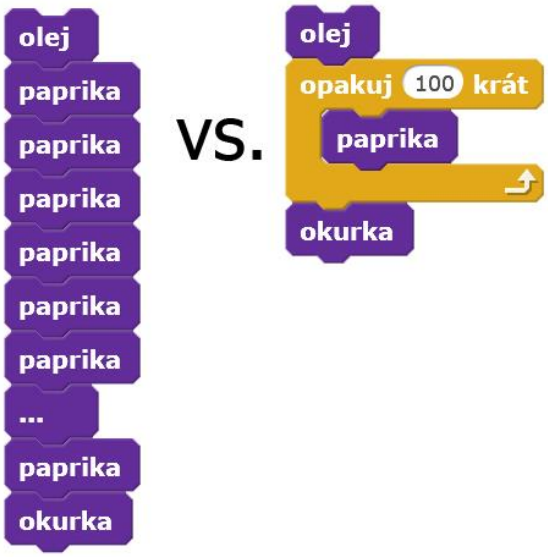


Obrázek 2.5 – příklad úlohy s využitím příkazu dopředu (Picka, 2018).



Obrázek 2.6 – příklad úlohy, která umožňuje různá řešení (Picka, 2018).

Pro výuku složitějších pojmů, jako například opakování (cyklus), podprogramy, rozhodování s podmínkou, práce s událostmi či komunikaci mezi objekty lze využít např. program Scratch.

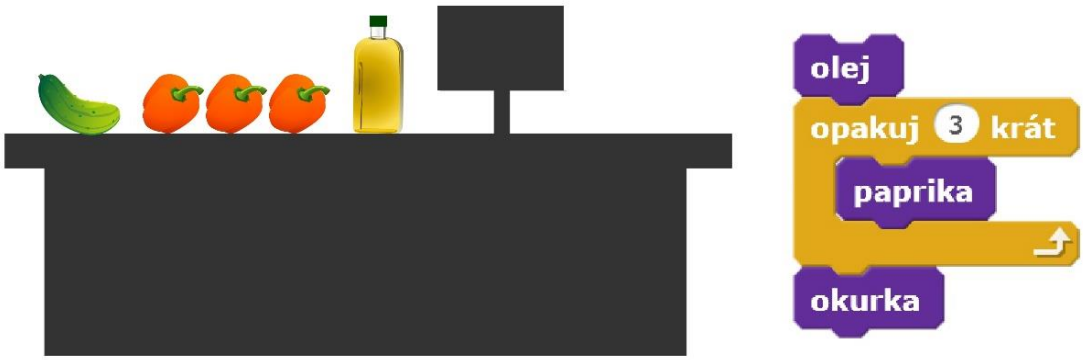


Obrázek 2.7 – řešení úlohy – vlevo pomocí řazení příkazů; vpravo pomocí cyklů.

Opakování (cykly)

Opakování neboli cykly tvoří jednu ze základních programátorských konstrukcí. Opakující se příkazy můžeme vložit dovnitř bloku opakování a určit kolikrát se mají příkazy uvnitř cyklu vykonat. Díky tomu můžeme dosáhnout menšího počtu zapsaných příkazů. Zkuste si představit, že máte nakoupit pro letní dětský tábor: jeden olej, sto paprik a jednu okurku. Z obrázku 2.7 je zřejmé, že vyřešení zadání pouze pomocí řazení příkazů by bylo oproti variantě s využitím cyklů časově náročnější, výsledný scénář by byl mnohem delší, méně přehledný a obtížněji modifikovatelný.

Pro správné pochopení cyklů je důležité rozeznat, které bloky jsou uvnitř opakování (vykonají se vícekrát), a které se nacházejí mimo blok opakování a tudíž se vykonají pouze jedenkrát.

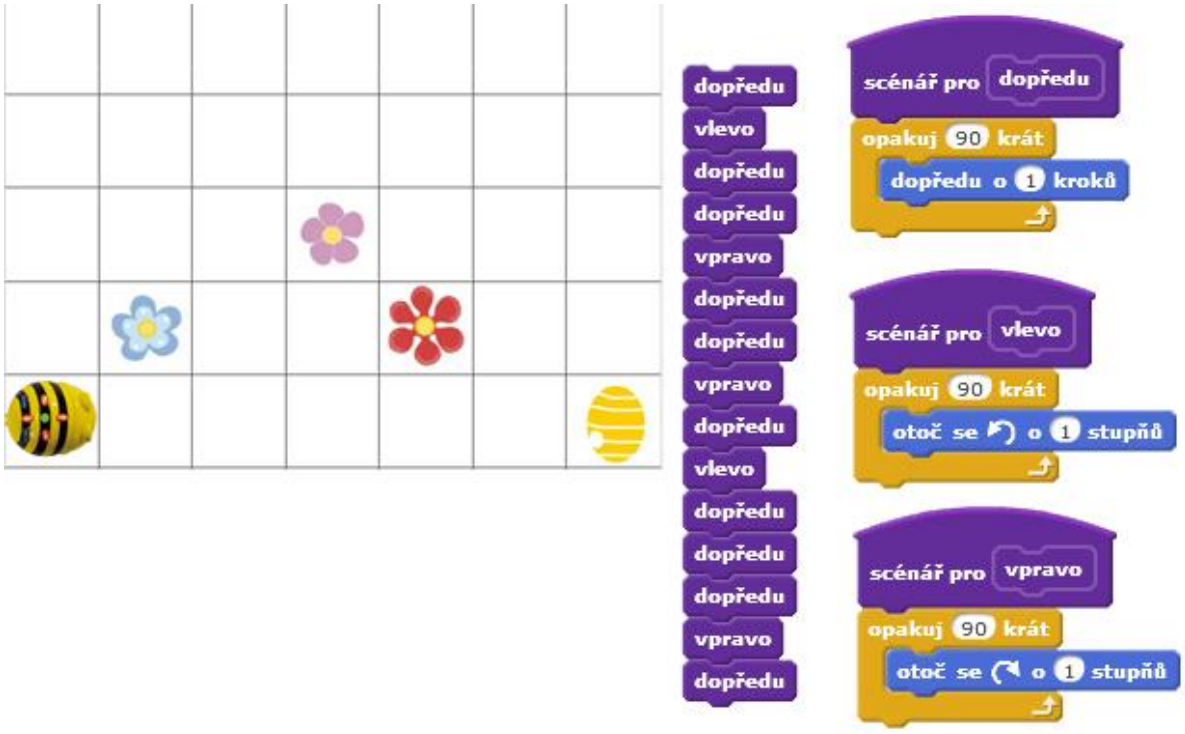


Obrázek 2.8 – příklad úlohy využívající blok opakování - je nákup na pokladním páse vyskládán dle scénáře vpravo?

Obrázek 2.9 – příklad úlohy využívající blok opakování –vyskládej nákup na pokladní pás podle scénáře vpravo.

Podprogramy

Mnohdy je třeba nějakou činnost provádět vícenásobně a na různých místech scénáře. V takovém případě je vhodné ji zapsat jako podprogram. Tímto způsobem lze snížit počet bloků, zlepšit srozumitelnost scénáře. Snazší může být ladění i případná modifikace. Využití podprogramu je znázorněno na obrázku 2.10, simulující jízdu reálně včelky Bee-bot v programu Scratch. Žák má za úkol navrhnout pro včelku takovou cestu do úlu, aby po cestě opilovala všechny květinky. Pokud včelka pojede dle scénáře vpravo, kudy pojede? Bylo by možné cestu včelce zkrátit?



Obrázek 2.10 – příklad úlohy využívající podprogramů.

Podmínky

Podmínka je vlastně otázka, na kterou program odpoví buď ano (podmínka splněná) nebo ne (podmínka nesplněná) a na základě této odpovědi se rozhoduje, co bude dělat dál.

Některé podmínky pro rozhodování jsou žákům 1. stupně srozumitelné (např. když se postava dotýká kurzoru myši, když stiskneme klávesu nebo když stojí na nějaké barvě). Podmínky, používající porovnání čísel nebo textů či dokonce proměnné (např. když délka = 10, když souřadnice x > 0) jsou pro tento věk žáka nepřiměřeně obtížné (Vaníček, 2018).

V případě ukázky na obrázku 2.11 Anička při stisku klávesy mezerník rozsvítí baterku. Pokud klávesa mezerník není stisknutá, baterka nesvítí. Správným

stiskem/podržením klávesy mezerník může Anička vyslat např. celosvětově rozšířený signál pro nouzové volání o pomoc – SOS.

Obrázek 2.11 – příklad úlohy využívající podmínku.

Spuštění scénáře událostmi

Každý scénář ve Scratchi musí vždy začínat nějakou událostí. Jedná se například o událost „PO KLIKNUTÍ NA VLAJKU“, nebo „PO STISKU KLÁVESY“ apod. Za událost se vždy připojuje jeden či více příkazů, které se začnou vykonávat právě ve chvíli, kdy daná událost v programu nastane (když se program spustí kliknutím na vlajku, když někdo zmáčkne příslušnou klávesu).

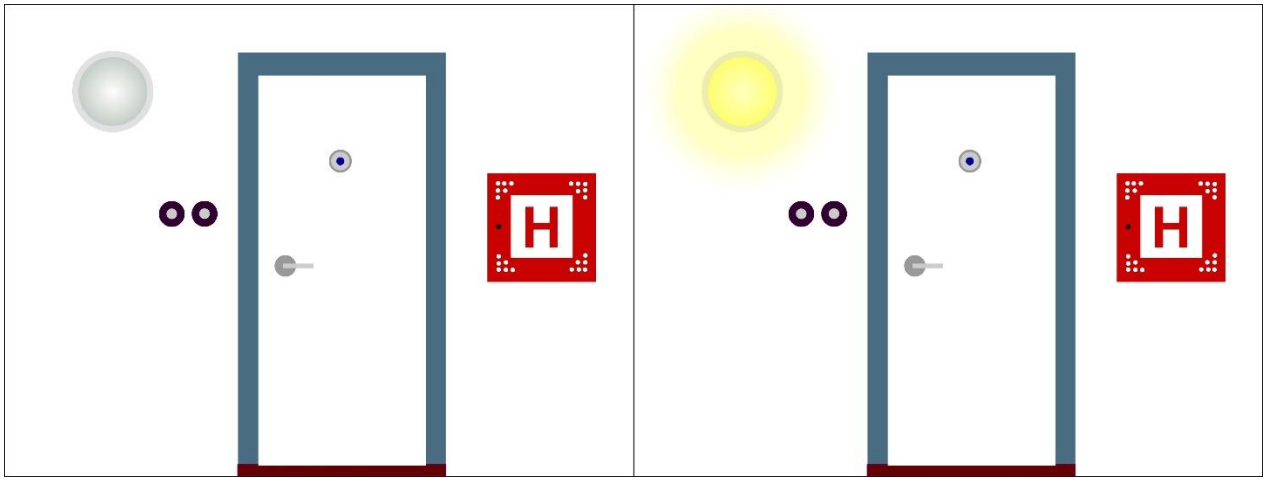
Obrázek 2.12 – ukázka vybraných bloků událostí.

Díky událostem je možné (a poměrně obvyklé), že v jednom programu běží několik scénářů najednou.

Komunikace mezi objekty

Vedle výše uvedených událostí umožňuje Scratch vytvářet také vlastní události, takzvané „Zprávy“. Jakýkoliv objekt může poslat libovolnou zprávu (určenou svým názvem) jinému objektu/objektům. Ostatní objekty mohou poslouchat, zdali některý z objektů neposlal zprávu s očekávaným názvem. Pokud ano, mohou na ni reagovat. Jednoduchým příkladem, využívajícím komunikaci mezi objekty, může být prostor chodby s osvětlením a schodišťovým spínačem.

Při kliknutí na objekt schodišťového spínače se objekt světlo musí na 10 vteřin rozsvítit a po uplynutí tohoto času opět zhasnout.



Obrázek 2.13 – příklad úlohy využívající komunikace mezi objekty. Vpravo – rozsvícená chodba po stisku schodišťového spínače.

Obrázek 2.14 – příklad scénářů k jednotlivým objektům.

2.3. Závěr

Základní porozumění zápisu algoritmů, čtení a opravy chyb u programů může přispět k rozvoji rozumových schopností dítěte. V pre-primárním vzdělávání je vhodné využívat elementárních programovacích jazyků, jednoduchých

robotických hraček, které dokážou podpořit prostorovou orientaci a představivost, tvořivost a paměť dítěte. V tomto studijním materiálu jsou popsány, některé z programovacích prostředí a robotických hraček. Velmi populární hračkou je Bee-bot. Hračka určená pro děti předškolního a mladšího školního věku. Robotické systémy rozvíjí algoritmické myšlení dětí, matematické představy a slovní zásobu dětí.

Literatura

Arrow Jumping Game [online] Discoveredby Tanjo [11-11-2015]. [cit. 06-08-2018]. Dostupné z: https://tanjo.ai/contents/448931

Bromová, J. Výuka algoritmizace na ZŠ – aktuální stav [online]. 2012. [cit. 19-06-2018]. Dostupné z: https://theses.cz/id/p527u4/DP_Bromova.pdf

Maněnová, M., Pekárková S. Rozvoj IM s využitím robotických hraček v MŠ a na 1. stupni ZŠ [online]. 2018. [cit. 29-07-2018]. Dostupné z: https://imysleni.cz/ucebnice/rozvoj-informatickeho-mysleni-s-vyuzitim-robotickych-hracek-v-materske-skole-a-na-1-stupni-zs

PICKA, Karel, 2018. *Robotické hračky 1 .stupeň*. Brno.

VANÍČEK, Jiří, 2018. *DITN – didaktika informačních technologií pro 1. stupeň ZŠ: 1, přednáška*. Jihočeská univerzita.

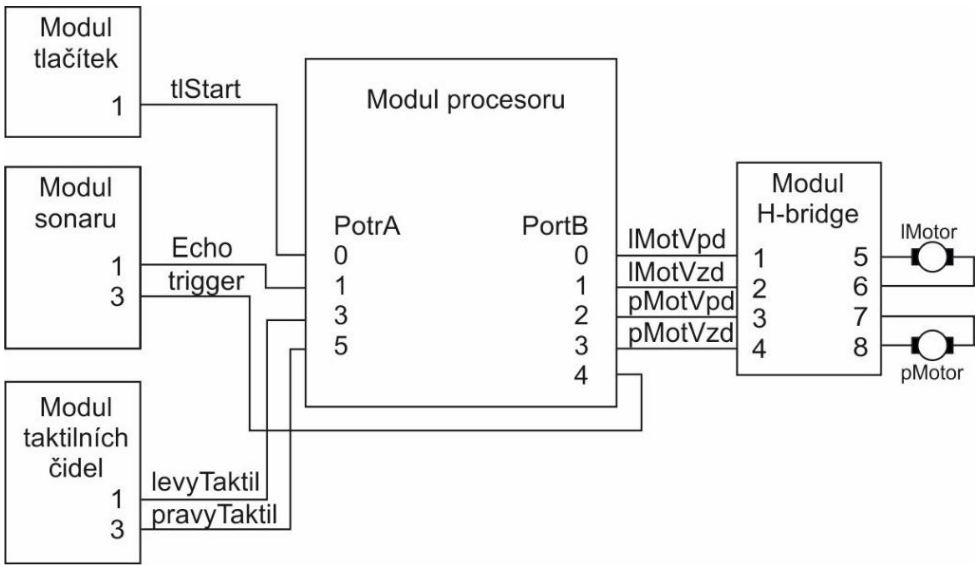
3. Základy práce s robotickými stavebnicemi a hračkami

Základy práce s robotickými stavebnicemi se skládají ze tří kroků, které od sebe nelze oddělit. Jde o návrh zapojení robotického systému a jeho propojení. Následuje oživení jednotlivých částí tohoto systému a posledním krokem je vytvoření programu, jeho odladění a na programování robota tak, aby dělal přesně to, co požadujeme. Těmto třem krokům se nyní budeme věnovat podrobněji.

3.1. Sestavení a oživení robota

Sestavení robota a vytvoření signálového schématu

Podle zadání je třeba nejdříve navrhnout, jaké části bude muset robot obsahovat (vstupní a výstupní moduly). Propojíme vstupy a výstupy modulu procesoru se vstupními a výstupními moduly robotického systému. Vstupní moduly obsahují senzory robota, ovládací tlačítka apod. Výstupní moduly pak jeho výkonné systémy, jako jsou motory a servosystémy apod. Všechny moduly připojíme k napájecímu zdroji. Toto propojení zakreslíme do signálového schématu (Obrázek 3.1). V něm označíme všechny signály (jejich symboly) a označení pinů (konektorů) kam jsou přesně připojeny. Do tohoto schématu obvykle nezakreslujeme připojení k napájecímu zdroji, protože každý modul musí být k odpovídajícímu napájecímu napětí připojen.



Obrázek 3.1 – Příklad signálového schémata.

Signálové schéma je důležité. Obsahuje přesné propojení jednotlivých modulů, označení signálů, které jsou v systému použity a které později při programování poslouží k vytvoření tabulky symbolů programu. Symbolické pojmenování signálů umožní intuitivnější psaní programu a snadnou orientaci v něm.

Signálové schéma také umožní (v budoucnu) znovu robota sestavit a naprogramovat již vytvořeným programem, nebo program modifikovat a podle potřeby upravit.

Oživení a základní diagnostika systémů robota

Nebudeme se nyní soustředit na Lego roboty, protože zde je ověření funkčnosti a oživení systému postaveného robota omezeno pouze na vytvoření jednoduchých diagnostických programů.

Pokud jde o ostatní robotické systémy, je vhodné před vlastním programováním, po sestavení a propojení robota ověřit funkčnost vstupních a výstupních modulů a systémů, včetně připojení propojovacích kabelů. Pokud bychom to neudělali, mohlo by se nám stát, že bychom začali psát program a program by nám nereagoval tak, jak bychom předpokládali. Chyby by však nebyly v programu, ale způsobilo by je právě špatné propojení nebo špatná funkčnost některého z dílu systémů robota.

Proto se zde na oživení jednoduchých systémů podíváme trochu blíže. Ovšem soustředíme se v této chvíli pouze na nejjednodušší vstupní a výstupní systémy robotů.

Z toho, co již o programování robotů v této chvíli víme, vyplývá, že ten kdo má zájem a chuť programovat a vytvářet robotické systémy, musí znát nejenom programování těchto robotů, ale měl by se orientovat také v tom, jak pracují jednotlivé vstupní a výstupní moduly tohoto systému a jak se s nimi pracuje.

Popis činnosti programu nejčastěji vyjadřujeme pomocí tzv. vývojových diagramů. Vývojový diagram je vlastně blokové vyjádření činnosti, ze kterého je na první pohled patrno, jak jednotlivé činnosti programu na sebe navazují a jak se program chová, reaguje-li na nějakou podmínku. Existuje norma, jak se „kreslí“ které části programu ve vývojovém diagramu.

Pro naše potřeby si tuto normu výrazně zjednodušíme a budeme používat jen několik málo bloků:

- Do oválu budeme psát pojmenování programu, nebo pokud sem napíšeme START, je to přechod do programu po zapnutí napájení robota.
- Do obdélníků budeme psát příkazy, nebo činnosti, které se mají postupně vykonat v daném pořadí.

- Do kosočtverce píšeme podmínku větvení programu. Při splnění podmínky pokračujeme větví bez kolečka. Větev s kolečkem je větev, kterou pokračujeme, není-li podmínka splněna.
- Šipky ve vývojovém diagramu ukazují na následující blok činnosti. Například na následujícím obrázku (Obrázek 3.2 – Test vstupních čidel) Po zapnutí napájení robota jeho procesor testuje stav vstupu logické sondy. Je-li v L, podmínka je splněna a je zhasnuta LED. Je-li v H, podmínka není splněna a LED se tedy rozsvítí. Poté program přechází opět na test podmínky, zda je stav vstupu logické sondy v L.

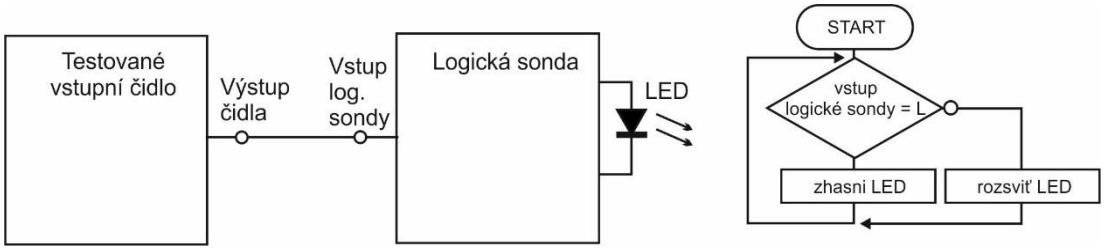
Nebudeme se do hloubky zabývat konstrukcí modulů a jejich přesnou činností. Je však pro nás důležité abychom věděli, jak tyto vstupní / vstupní moduly budou konkrétně spolupracovat s procesorem robota.

Nejužívanějším modulem je obvykle modul tlačítek. Tlačítka umožní uživateli robota spustit a popřípadě jej pomoci tlačítek ovládat.

Vstupní moduly – vstupní čidla

Vstupem **modulu tlačítek** je stisknutí nebo uvolnění tlačítka a výstupem je logická úroveň odpovídající stisknutému nebo uvolněnému tlačítku. Podle logické úrovně tedy systém pozná, zda je tlačítko stisknuté nebo uvolněné.

Pokud neznáme zapojení modulu, budeme na ně pohlížet jako na „černou skříňku“. Abychom zjistili, jak funguje, vytvoříme si velmi jednoduchou logickou sondu z robota.



Obrázek 3.2 – Test vstupních čidel.

Jde-li o tlačítka, pak vstupem modulu tlačítek je stisk tlačítka. Výstupem je logická úroveň. Logická úroveň L je logická 0, Low, není pravda. Logická úroveň H je logická 1, High, je pravda.

Mohou nastat dvě situace.

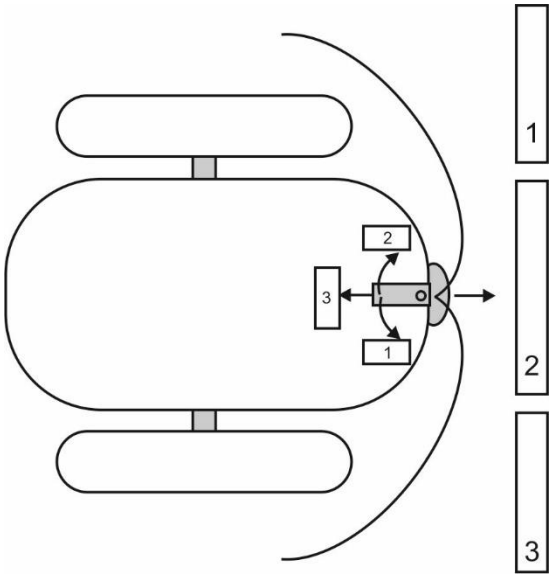
První z nich: Není-li stisknuté tlačítko, výstupem modulu tlačítek je L. Při stisknutém tlačítku je výstup v H. Logická úroveň výstupu znamená následující:

Výstup v L říká, že není pravda, že je stisknuté tlačítko. Výstup v H pak znamená, že je pravda, že je stisknuté tlačítko.

Druhá: Není-li stisknuté tlačítko, výstupem modulu tlačítek je H. Při stisknutém tlačítku je výstup v L. Logická úroveň výstupu znamená následující:

Výstup v H říká,že je pravda, že není stisknuté tlačítko. Výstup v L pak znamená, že není pravda, že není stisknuté tlačítko.

Podobně jako tlačítka v modulu tlačítek pracují také **taktilní čidla** (dotyková čidla), která v nejjednodušším případě mohou být speciálně upravenými tlačítky. Taktilní čidlo je opatřeno jakýmsi „tykadlem“. Dojde-li k dotyku tohoto „tykadla“ ve správném směru, má to stejný efekt, jako stisk tlačítka. Například při nárazu na překážku 1 levým tykadlem, ramínko taktilního čidla se nakloní ve směru šipky směrem k tlačítku 1, které sepne. Při čelním nárazu na překážku 2 oběma tykadly ramínko taktilního čidla zatlačí na tlačítko 3 a sepne jej. Při nárazu pravým tykadlem na překážku 3, ramínko taktilního čidla se nakloní ve směru šipky a sepne tlačítko 2, viz Obrázek 3.3.



Obrázek 3.3 – Příklad taktilních čidel na robotovi.

Vstupem taktilního čidla je dotyk čidla (zda došlo, nebo nedošlo, k dotyku taktilního čidla). Výstupem je logická úroveň odpovídající situaci, kdy k dotyku došlo a kdy k dotyku nedošlo. Podle logické úrovně tedy systém pozná, zda k dotyku došlo, nebo nikoliv.

Mohou opět nastat tyto dvě situace.

První z nich: Nedošlo-li k dotyku taktilního čidla s překážkou, výstupem taktilního čidla je L. Došlo-li k dotyku taktilního čidla s překážkou, výstupem taktilního čidla je H. Logická úroveň výstupu znamená následující:

Výstup v L říká,že není pravda, že došlo k dotyku. Výstup v H pak znamená, že je pravda, že došlo k dotyku.

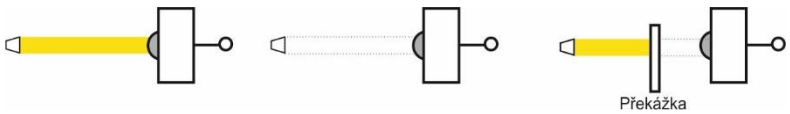
Druhá: Nedošlo-li k dotyku taktilního čidla s překážkou, výstupem taktilního čidla je H. Došlo-li k dotyku taktilního čidla s překážkou, výstupem taktilního čidla je L. Logická úroveň výstupu znamená následující:

Výstup v H říká, že je pravda, že nedošlo k dotyku. Výstup v L pak znamená, že není pravda, že nedošlo k dotyku.

Posledními vstupními moduly, kterými se budeme v této kapitole zabývat, jsou moduly citlivé na světlo nebo přesněji na infračervené záření (neviditelné záření)takzvané **infra senzory**.

Základním prvkem těchto infra senzorů jsou tak zvané fototranzistory, které se chovají jako tlačítka řízená právě tímto infračerveným zářením.

Svítí-li na fototranzistor infračervené záření, fototranzistor se podobně jako tlačítko sepne. Pokud na tento fototranzistor nedopadá infrazáření pak je podobně jako tlačítko rozepnutý.



Obrázek 3.4 –Transmisní optické infračidlo.

Na obrázku vlevo na fototranzistor světlo/infrazáření dopadá – čidlo je osvětlené. Prostřední a pravý obrázek ukazuje, že na čidlo záření nedopadá.

Může to být způsobeno neprůhlednou překážkou, nebo přestal záření vysílat jeho zdroj. Takto pracuje takzvané **transmisní optické čidlo**, viz Obrázek 3.4.

S takovýmto čidlem se můžeme setkat například u výtahu, kdy nám čidlo hlídá, aby po uzavření kabiny nikdo nestál v prostoru dveří.

Druhým typem optického čidla jsou **čidla reflexní** (odrazivá), viz Obrázek 3.5. Zde vysílač světelného záření svítí směrem od čidla. V okamžiku, kdy se objeví překážka, která odrazí toto světlo zpět do čidla na fototranzistor, tento sepne. Je to podobné, jako když vezmeme zrcátko a pomocí zrcátka vrháme „prasátka“.

Obrázek 3.5 – Činnost reflexního čidla.

Tato čidla se využívají například k tomu, aby robot jel podle čáry. Dopadá-li infra záření na čáru ta jej pohltí a neodrazí do fototranzistoru. Pokud robot sjede z čáry a infra záření dopadá na plochu kolem čáry, ta jej odrazí zpět do fototranzistoru.

Mohou opět nastat dvě situace.

První z nich: Dopadá-li záření na čidlo, výstupem čidla je L, nedopadá-li, pak je výstup v H. Logická úroveň výstupu znamená následující:

Výstup v L říká, že je pravda, že je čidlo osvětlené. Výstup v H pak znamená, že není pravda, že čidlo je osvětlené.

Druhá: Dopadá-li záření na čidlo, výstupem čidla je H, nedopadá-li, pak je výstup v L. Logická úroveň výstupu znamená následující:

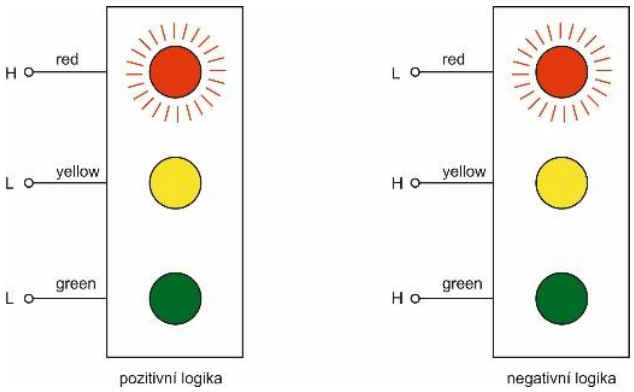
Výstup v H říká, že je pravda, že je čidlo osvětlené. Výstup v L pak znamená, že není pravda, že čidlo je osvětlené.

Výstupní moduly

Nejčastější výstupní moduly (zařízení)využívané u robotických systémů jsou indikační moduly (moduly se svítivými diodami LED), tranzistorové spínače pro připojení motorů, Laserů, sirén apod. Nesmíme zapomenout ani na H-bridge (H-mosty) pro připojení motorů. Motor připojený k tranzistorovému spínači může buď jet, nebo ne (siréna houká/nehouká, laser svítí/nesvítí). Motor připojený k H-mostu může jet na jednu stranu/na druhou stranu, může být vypnuto jeho napájení – zastavení bez brzdění, nebo zastavit s brzděním. V této chvíli mluvíme o běžných komutátorových stejnosměrných motorech připojených k H-mostu.

Indikační moduly

Pomocí LED diod (svítivých diod) můžeme indikovat různé situace a stavy systému. Nebezpečí, bezchybnou funkci, stav zda došlo k nějaké situaci, zda například běží čerpadlo, topí topení, zařízení je zapnuté, nebo vypnuté apod. Dokonce i například semafory do této skupiny patří. Jde tedy o indikační prvky. Vstupem těchto prvků je logická úroveň na jejich vstupu. Výstupem pak svit příslušné LED.



Obrázek 3.6 – Ovládání LED indikace - pozitivní a negativní logika.

Abychom mohli tyto prvky ovládat, musíme znát, jaká úroveň signálu na vstupu způsobí, že prvek svítí a jaká že nesvítí.

Jak vidíme, stejně jako u vstupních zařízení i zde mohou nastat dvě situace, viz Obrázek 3.6.

První z nich - **pozitivní logika**: Přijde-li na vstup výstupního indikačního obvodu s LED diodou logická úroveň H, výstupem bude svit LED (Led se

rozsvítí). Přijde-li na vstup výstupního indikačního obvodu s LED diodou logická úroveň L, výstupem nebude svit LED (LED zhasne).

Vstup v L říká, že není pravda, že LED svítí. Vstup v H pak znamená, že je pravda, že LED svítí.

Druhá z nich - **negativní logika**: Přijde-li na vstup výstupního indikačního obvodu s LED diodou logická úroveň H, výstupem nebude svit LED (LED zhasne). Přijde-li na vstup výstupního indikačního obvodu s LED diodou logická úroveň L, výstupem bude svit LED (LED se rozsvítí).

Vstup v L říká, že není pravda, že LED nesvítí. Vstup v H pak znamená, že je pravda, že LED nesvítí.

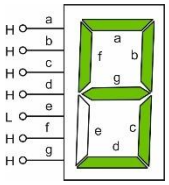
Na těchto příkladech vidíme, že význam logických signálů může být dvojí. Je to způsobeno tím, jak logické obvody pracují, jak jsou navrženy. Logické obvody mohu tedy pracovat s pozitivní, nebo negativní logikou. Oba typy logiky ve svém důsledku fungují zcela stejně. Liší se pouze v chování na logickou úroveň L a H. Jak již víme, H, znamená logickou jedničku, vysokou úroveň (High), nebo také TRUE – pravdu. L, znamená logickou nulu, nízkou úroveň signálu (Low), nebo také FALSE – není pravda (nepravdu).

Pozitivní logika: Vstup v H znamená, že je pravda, že LED svítí.

Negativní logika: Vstup v L pak znamená, že není pravda, že LED nesvítí.

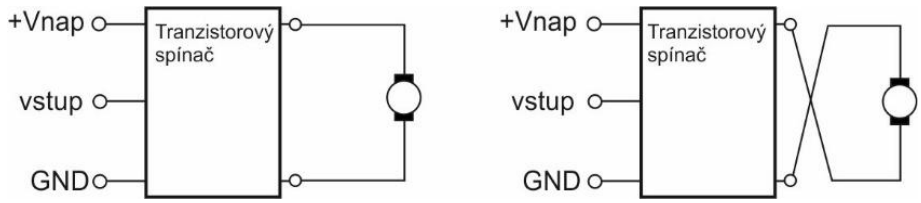
Vidíme, že oba výroky definující činnost mají stejný význam. ***Je pravda, že LED svítí*** je totéž jako ***není pravda, že LED nesvítí.*** V pozitivní logice se LED rozsvítí pomocí H, v negativní logice pak pomocí L. My už budeme pro jednoduchost další výklad vést s využitím pozitivní logiky, což je pro nás názornější.

Mezi indikační výstupní zařízení se dají také zařadit – například sedmi segmentové číslicové zobrazovače, které mají 7 vstupů a 7 LED segmentů. Jejich kombinace mohou například indikovat číslice 0 – 9, viz Obrázek 3.7.



Obrázek 3.7 – Příklad sedmisegmentové zobrazovací jednotky.

Tranzistorový spínač

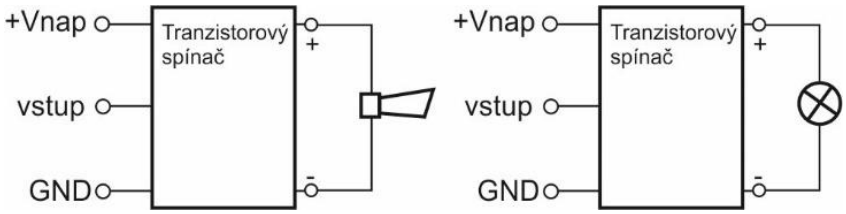


Obrázek 3.8 – Příklad signálového schémata.

Vstup v H pak znamená, že je pravda, že motor běží. Vstup v L říká, že není pravda, že motor běží.

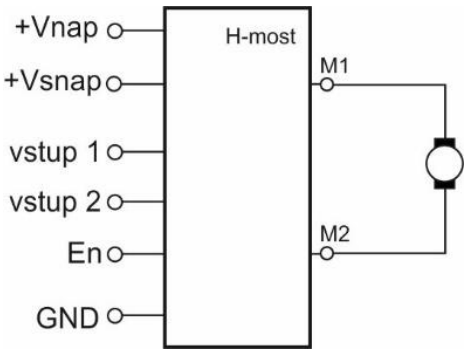
Tento spínač se používá ke spínání komutátorových motorů (běžné motory používané v různých hračkách a modelech). Změna směru otáčení motoru se dá dosáhnout prohozením vývodů motoru k tranzistorovému spínači, jak je patrné z Obrázku 3.8. Vstupem je tedy logická úroveň signálu (H/L), výstupem pak otáčení motoru.

Tranzistorový spínač můžeme bez problémů použít například pro připojení výkonové sirény, nebo reflektoru. V daném případě připojíme zařízení na místo připojeného motoru, viz Obrázek 3.9.



Obrázek 3.9 – Připojení sirény a reflektoru k tranzistorovému spínači.

H-Bridge (H-most)



Obrázek 3.10 – Ovládání motoru pomocí H-bridge.

Tento výstupní modul se využívá pro řízení krokových motorů, nebo komutátorových motorů. Řízení krokových motorů se zde nebudeme věnovat, je nad naše možnosti a znalosti. Použijeme-li komutátorové motory, jako v předchozím případě, je toto řízení H-mostem velmi efektivní. Pomocí mostu můžeme řídit jeden motor vpřed, vzad, zastavení se setrvačností (odpojením od napájení), nebo zastavení s brzděním zkratem vývodů motoru připojením motoru vývodům M1 a M2. Motor můžeme napájet různými napětími a to až 36 V (vstup +Vnap). Obvod potřebuje pro svou práci také napájení +Vsnap napájení logických obvodů řídících most (+5V). Pro oba zdroje i řídící logické signály vstup1, vstup 2 i vstup EN je GND společnou zemí.

vstup1	vstup2	vstup EN	
0	0	1	off motoru s brzděním
0	1	1	jeden směr jízdy
1	0	1	opačný směr jízdy
1	1	1	off motoru s brzděním
x	x	0	off motoru

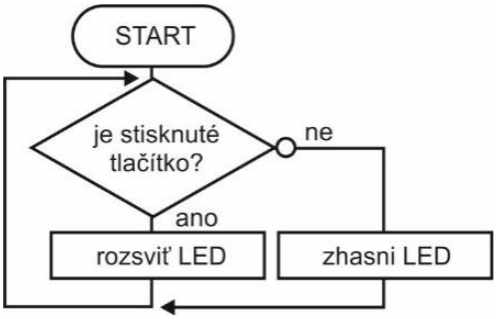
Diagnostické programy pro ověření funkčnosti

Před tím, než se pustíme do programování konkrétní aplikace, je nezbytné ověřit funkčnost všech částí robotického systému. Je tedy po návrhu a propojení celého systému nutné funkčnost ověřit. K tomu nám poslouží řada velmi jednoduchých programů, které jednotlivé části otestují. To umožní nalézt například špatné připojení motorů (nastavení jejich směru v souladu s řídícími signály), odhalení vadných vodičů, kterými jsou vstupy/výstupy propojeny s procesorem apod. Zásadní pro činnost programu je funkčnost tlačítek a vstupních čidel, podle kterých je celý systém řízen. Nefunguje-li správně některé z těchto čidel, činnost systému se zhroutí a v krajním případě může dojít k chybám, které mohou dokonce ohrozit životy lidí, nebo způsobit velké finanční ztráty (řídí-li systém například automobil, kosmickou raketu, nebo montážní linku v závodu).

Pokud diagnostiku neprovedeme před začátkem programování, nebo ve chvíli, kdy se přestane systém chovat správně, můžeme strávit mnoho zbytečných hodin hledáním chyb v řídícím programu systému.

Ukážeme si nyní příklady několika testovacích programů pro ověření činnosti výše popsaných modulů.

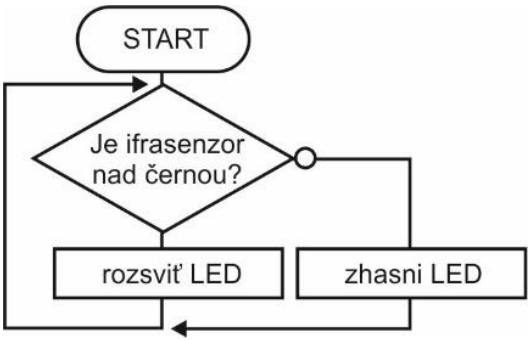
Test tlačítka a indikační LED



Obrázek 3.11 –Test tlačítka a LED.

Ve vývojových diagramech není třeba psát k větvím rozhodovacího bloku ano/ne. Větev s kolečkem je větev, kdy není splněna podmínka – je stisknuté tlačítko? Není – rozhodovací blok obsahuje otázku, na kterou je možné odpovědět pouze ano, nebo ne. Procesor robota po zapnutí (blok Start) testuje, zda je tlačítko stisknuté (výstup tlačítka H/L). Když není tlačítko stisknuté, program pokračuje větví s kolečkem – zhasne LED. Pak se vrací zpět na test tlačítka. Dokud nebude tlačítko stisknuté, běží program popsanou větví. Ve chvíli, kdy někdo stiskne tlačítko, změní se výstupní signál tlačítka. Až program přijde na test tlačítka, přejde do větve bez kolečka a rozsvítí LED. Musíme si uvědomit, že tento program po dobu stisknutého tlačítka LED cyklicky rozsvěcuje. Pro nás je to nepodstatné, protože když rozsvítíte již svítící LED, LED bude stále svítit. Podobné je to kdy není tlačítko stisknuté a program cyklicky vypíná LED, která nesvítí. Tyto činnosti dělá procesor až mnoho milionkrát za sekundu. První test tedy otestuje funkčnost indikace a tlačítek (Obrázek 3.11).

Podobně otestujeme infra snímač

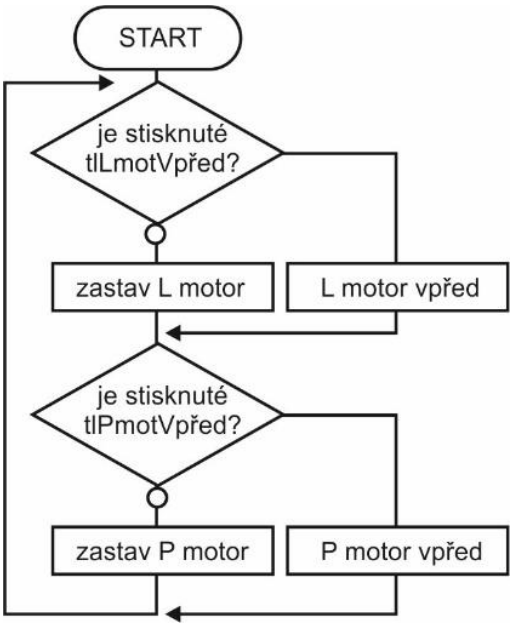


Obrázek 3.12 –Test infrasenzoru.

Pokud budeme využívat infrasenzor na indikaci – pak nachází-li se pod ním černá (pohlcující) plocha, nebo bílá (odrazivá) plocha například pro řízení jízdy robota po čáře, můžeme jeho funkčnost otestovat výše uvedeným způsobem.

Kdyby bylo okolní osvětlení robota vysoké (slunce), senzor by přestal indikovat, nad jakou plochou se nachází a robot by přestal sledovat čáru. Bez testovacího programu tuto situaci jen velmi těžko odhalíme (Obrázek 3.12).

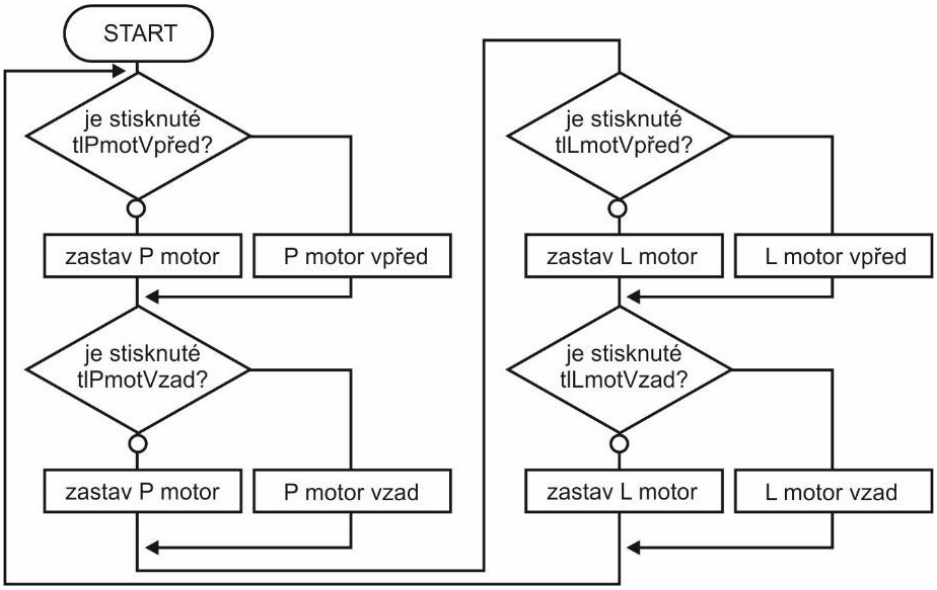
Test řízení motorů spínači



Obrázek 3.13 –Test tranzistorových spínačů s motory

Použijeme-li dvě tlačítka, můžeme snadno otestovat, zda jsme správně zapojili tranzistorové spínače s motory na správné vývody procesoru (levý a pravý) a současně poznáme, zda jsme správně připojili motory ke spínači a ty se otáčejí správným směrem. Pokud by tomu tak nebylo, můžeme výstupy signálů z robota připojit ke správným vstupům tranzistorových spínačů. Stejně tak můžeme správným způsobem připojit motory k výstupům tranzistorových spínačů tak, aby se otáčely správným směrem vpřed (Obrázek 3.13).

Test řízení motorů robota pomocí H-bridge



Obrázek 3.14 –Test spojení H-bridge s motory robota.

Nyní můžeme otestovat, zda máme správně propojené výstupní řídící signály procesu robota s odpovídajícími signály H-bridge a v souvislosti s tím správné propojení H-bridge s motory robota tak, aby správně reagovaly na řídící signály. Tuto kontrolu a nastavení nesmíme nikdy podcenit. Je zásadní pro řízení jízdy robota (Obrázek 3.14).

3.2. Vytváření programu pro robota

Programování je založeno na algoritmizaci postupu řešení nějakého zadaného problémů. Vycházíme ze zadání problém. Následuje rozložení daného problémy na jednotlivé parciální na sebe navazujících části - posloupnost příkazů nebo operací které vedou k vykonání daného zadání. Postupně se algoritmus stále víc zpřesňuje a přizpůsobuje možnostem systému (procesoru), který tyto úkoly má řešit. Posledním krokem na této cestě je přepsání jednotlivých kroků do konkrétních příkazů (instrukcí), který umí provádět řídící procesor systému, tedy naprogramování aplikace do systému.

Programovat lze pomocí grafických programovacích technik, kde jsou rozhodovací bloky a příkazy ve formě bloků. Rozepsaný vývojový diagram pomocí těchto bloků převedeme do konkrétního programu.

Pokud jednou začne procesor pracovat, nikdy se nemůže zastavit jeho činnost. START znamená zapnutí napájení procesoru, kdy se procesor probudí a začne provádět příkazy našeho programu. Pokud potřebujeme, aby nic nedělal a zastavil, musíme vytvořit smyčku, kde čeká na splnění podmínky (stisk tlačítka, příchod signálu od nějakého senzoru apod.), aby mohl program dále pracovat.

Můžeme si zde ukázat příklad takového jazyka, který je zacílen na mikrokontroléry PICAXE. Tento přístup je typický pro podobné programovací nástroje a nejbližší základním blokům vývojových diagramů.

První program - blikání LED

Propojíme napájení desky LED s vývody +5V a GND na desce procesoru. Propojíme OUT0 desky procesoru s vývodem 1 desky LED.

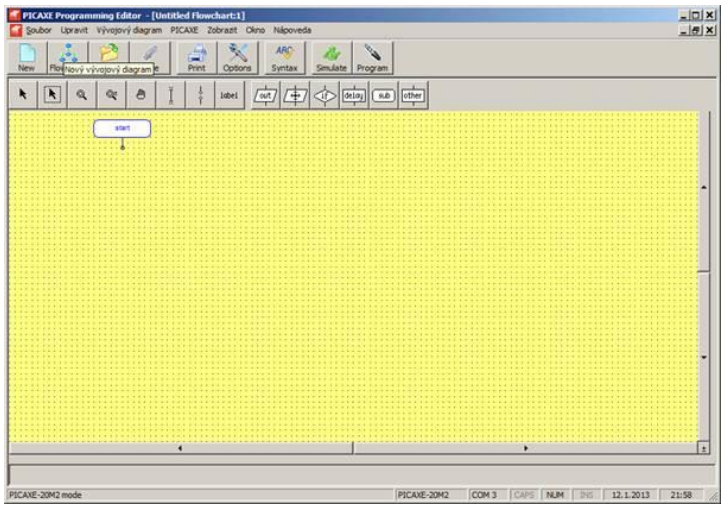
Blikání LED bude probíhat následovně - 1 s. svítí a 1 s. nesvítí

1. Rozsviť LED.
2. Čekej 1 s.
3. Zhasni LED.
4. Čekej 1 s.
5. Jdi na 1. Příkaz.

Vytvoření programu v programovacím editoru

Vytvoříme vývojový diagram, který bude grafickým vyjádřením výše popsanému postupu činnosti programu pro blikání LED v programovém editoru.

V editoru otevřeme Flowchart – vývojový diagram (dále už jen zjednodušeně, jak říkají programátoři „vývojáky“), (Obrázek 3.15 až 3.20).

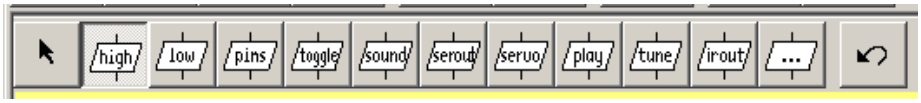


Obrázek 3.15 –Vývojové prostředí PICAXE Programming editor.

Nástroje hlavního menu

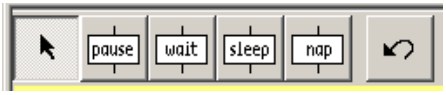
Obrázek 3.16 –Nástroje PICAXE Programming editoru

Nástroje „out“ - Slouží k řízení výstupů procesoru



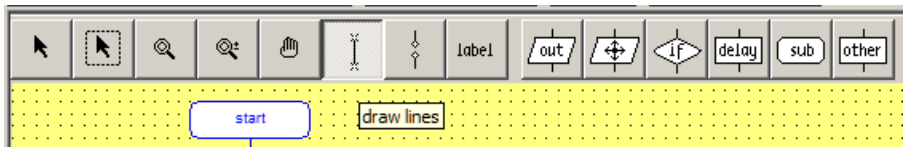
Obrázek 3.17 –Nástroje PICAXE Programming editoru pro výstupy.

Nástroje „delay“ – slouží k realizaci čekání



Obrázek 3.18 –Nástroje PICAXE Programming editoru pro čekání.

Nástroj pro propojení bloků – nástroj draw line



Obrázek 3.19 –Test spojení H-bridgů s motory robota.

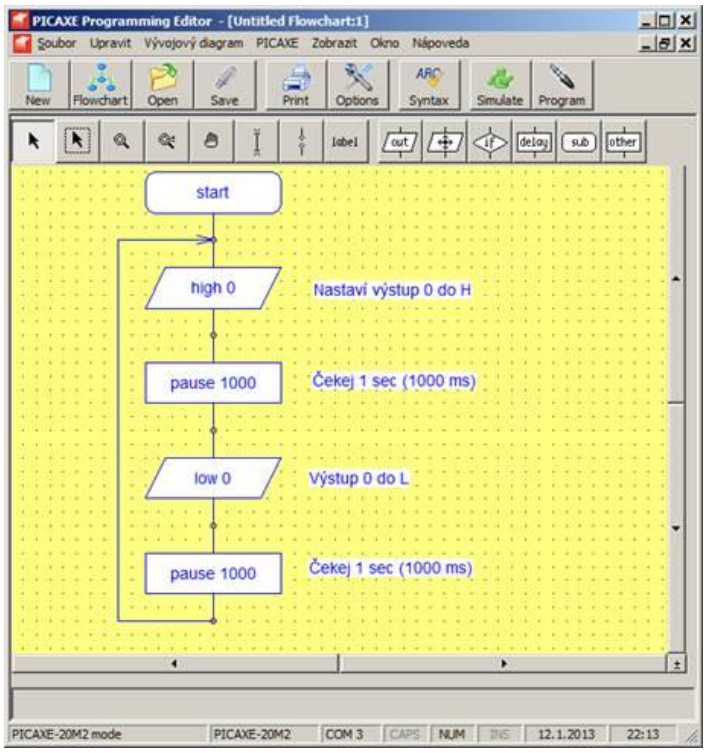
Použité instrukce (příkazy)

- v programu

start
high *cislo*
low *cislo*
pause *cislo*

Naprogramování procesoru

Nakonec program přeneseme do procesoru - procesor naprogramujeme. Po naprogramování začne procesor program automaticky provádět a LED 1 bude blikat v intervalu 1 s.



Obrázek 3.20 –Program blikání LED vytvořený v PICAXE Programming editoru.

3.3. Závěr

Nyní již máme k dispozici důležité základní informace pro návrh a programování robotických systémů. Záměrně je text psán tak, aby byl platný a nezávislý na typu robotické stavebnice, nebo procesoru robota. V této oblasti jde především o metodiku jak tyto systémy navrhovat a realizovat. Přizpůsobit tyto znalosti konkrétnímu systému je jen rutinní záležitostí. Nejdůležitější je způsob myšlení a postup návrhu.

Literatura

Ukázka práce s PICAXE programming editorem BAS805 [online]. © 2013 [cit. 2018-09-01]. Dostupné z WWW:< http://hses.cz/wp-content/uploads/2018/08/První-program.pdf >.

PICAXE manuals [online]. © 2013 [cit. 2018-09-01]. Dostupné z WWW:<http://www.picaxe.com/Getting-Started/PICAXE-Manuals/ >.