ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta pedagogická

Katedra výpočetní a didaktické techniky

Diplomová práce

APLIKACE PRO ÚVOD DO VÝUKY PROGRAMOVÁNÍ

David Žáček

Plzeň 2009

Prohlášení:

Prohlašuji, že jsem práci vypracoval samostatně s použitím uvedené literatury a zdrojů informací.

. . . . . .

Plzeň, 25. 3. 2009 David Žáček

Poděkování:

Rád bych poděkoval vedoucímu mé diplomové práce Doc. Ing.Václavu Vrbíkovi, CSc. za jeho zájem, cenné rady a připomínky.

Obsah

Kapitola 1

Úvod

Někteří ze studentů, kteří přicházejí studovat na střední školu, prošli během základní školy v rámci hodin výpočetní techniky nějakým kurzem programování, většinou v tzv. dětském jazyku, ale většina z nich nemá s programováním téměř žádné zkušenosti. V RVP pro obor vzdělávání Technické lyceum je mimo jiné stanoveno, že studenti mají ovládat základy programování ve vyšším programovacím jazyce, algoritmizace a OOP.

V mém povědomí je však většina kurzů objektově orientovaného programování spíše procedurální povahy, v podstatě vychází z dřívější výuky procedurálních jazyků a k objektům se studenti dostanou až na konec. Navíc pro studenty, kteří se s programováním ještě nesetkali, bývá výuka mnohdy náročná a někdy zbytečně demotivující. Zdá se mi smysluplné hledat způsob, jak jim úvod do objektově orientovaného programování zpříjemnit, zjednodušit a ulehčit jim jejich první kroky snížením stupně abstrakce. Smysluplné je tedy najít chybějící mezičlánek použitelný na úvod výuky, který bude podobný dětským prostředím používaných na základních školách, např. Logo nebo Baltík, ale na úrovni potřebné pro úvod do programování na SŠ.

Napadlo mě tedy vytvořit jednoduché a přehledné prostředí, umožňující intuitivní ovládání na dotykové tabuli. Aplikaci, která by byla inspirována známým robotem Karlem. Výuka pomocí této aplikace by se měla orientovat na řešení problémů při vytváření algoritmů, pochopení principů a uvědomění si souvislostí. Začátek by měl být co nejintuitivnější a studenti by měli po malinkých krůčcích postupovat dál. Budou tedy programovat robota, který má dané instrukce a plní úkoly. Robot bude ze začátku znát jen pár svých instrukcí a studenti nebudou kód psát, jednotlivé instrukce si budou vybírat z nabídky. Vytvořený program si budou moci tzv. odkrokovat a dívat se, zda-li robot opravdu udělá to, co měl.

Studenti tak nebudou zahlceni informacemi, nemusí se zabývat syntaxí a budou se více orientovat na řešení problému a vytváření algoritmu. Na rozdíl od robota Karla ale použijeme syntaxi vyššího programovacího jazyka, zaměříme se na osvojování základních principů algoritmizace úloh a na úvodní seznámení s objekty.

Cílem mé diplomové práce je tedy vytvoření načrtnuté aplikace vhodné pro úvod do výuky programování na střední škole, včetně navržení vhodné sady úloh a vy-

pracování metodických pokynů pro práci s nimi. Úlohy budou určeny pro učitele, který s jejich pomocí může připravit vyučovací hodiny a stanovit konkrétní cíle tak, aby studenti postupně po malých krůčcích řešením úkolů vstupovali do světa algoritmizace, objevovali možnosti základních programových konstrukcí jazyka Java a seznámili se s některými principy OOP.

V diplomové práci se na začátku zabývám hlavně tím, proč a jak použít na úvod do programování program R.U.R.[1] Rozebírám jeho možné výhody a nevýhody, poté následuje metodický popis úloh, včetně stanovených cílů a poznámek k jejich naplňování.

Třetí kapitola obsahuje uživatelskou příručku s přehlednými návody pro práci s aplikací. Ve čtvrté kapitole jsou nastíněny možnosti modifikace.

[1]Název programu R.U.R byl rovněž zvolen jako pocta k 70. výročí umrtí Karla Čapka, který společně se svým bratrem vymyslel celosvětově známé slovo robot. Toto slovo se poprvé objevilo právě v jeho divadelní hře R.U.R. (Rossum's Universal Robots).

Kapitola 2

Úvod do programování pomocí programu R.U.R

2.1 Cíle stanovené RVP

Cíle vychází z rámcového vzdělávacího programu pro obor vzdělávání Technické lyceum [2]. Kde je mimo jiné stanoveno, že vzdělávání v rámci odborných kompetencí má směřovat také k tomu, aby absolventi ovládali algoritmizaci úloh a základy programování ve vyšším programovacím jazyce, řešili jednodušší programátorské úlohy.

Kurikulární rámec vymezuje závazný obsah v oblasti vzdělávání v informačních a komunikačních technologiích v sekci Programování tak, aby naplňoval takto stanovené cíle:

- Ovládá principy algoritmizace úloh a je schopen sestavit algoritmus řešení konkrétní úlohy (dekompozice úlohy na jednotlivé elementárnější činnosti za použití přiměřené míry abstrakce).
- Vytvoří a odladí jednoduchý program v některém vývojovém prostředí.
- Ovládá základy objektově orientovaného programování.

2.2 Proč pro úvod do výuky programování používat program R.U.R?

Aplikace R.U.R. je primárně určena pro úvod do výuky programování na středních školách, případně na VŠ. Pro přímé užití ve výuce je připravena, a dále přehledně popsána, sada úloh s metodickými pokyny. Jako vyšší programovací jazyk byl zvolen moderní a rozšířený objektově orientovaný jazyk JAVA.

Sada úloh není učebnicí, je pouze metodickou pomůckou pro učitele, který si s její pomocí sám připravuje vyučovací hodiny a stanovuje konkrétní cíle. Kurz vytvořený učitelem se pak pro studenty může stát snadněji pochopitelným a ne tak náročným jako většina klasických kurzů programování.

Studenti se s novými pojmy a principy seznamují postupně v jednotlivých úlohách formou hry. Programují chování robota, takže jakákoliv akce je viditelná, čímž se snižuje stupeň abstrakce. Studenti nejsou zahlcováni informacemi a neprojeví se tak blok daný domnělou složitostí jazyku JAVA. Naopak s poměrně omezenými možnostmi se studenti snaží vyřešit přiměřeně náročný a jasně definovaný problém. Zažívají úspěch a jsou přirozeně motivováni k řešení dalších úloh a postupnému osvojování nových pojmů, konstrukcí a principů, které jim umožní daný problém úspěšně vyřešit. Bohatost jazyka postupně vnímají jako výhodu.

Výuka pomocí této aplikace je orientována na řešení problémů při vytváření algoritmů, pochopení principů, uvědomění si souvislostí. Učí studenty myslet, ale v podstatě se studenti nenaučí obecně programovat a je tedy nutné dále pokračovat v nějakém vhodném vývojovém prostředí.

Součástí diplomové práce je také uživatelská příručka obsahující přehledné návody pro práci s aplikací.

2.2.1 Výhody a nevýhody

Téměř všechny výhody po čase přecházejí v nevýhody. V příhodný čas je tedy vhodné přejít do jiného, pro naplňování cílů vhodnějšího, vývojového prostředí, aby studenti nezískali nevhodné návyky. Aplikace a navržené úlohy jsou zamýšleny jen jako úvod do programování. Nemohou nahradit celý kurz programování v JAVĚ.

Výhody:

- Studenti se s novými pojmy a principy seznamují formou hry.
- Velmi jednoduché prostředí.
- Skladba navržených úloh podporuje intuitivní a časově nenáročné seznámení s prosředím a jeho ovládáním.
- Omezené možnosti, studenti nejsou zahlceni informacemi.
- Orientace na řešení problémů při vytváření algoritmů, pochopení principů, uvědomění si souvislostí.
- Studenti se nemusí přímo zabývat syntaxí – zdržovat se psaním kódu.
- Programování v této aplikaci je relativně rychlé.
- Jakákoliv akce je viditelná, nižší stupeň abstrakce.
- Jasně viditelná, definovaná chyba – výbuch robota, a správný výsledek – robot udělá to, co se od něj očekává.
- Studenti se naučí krokovat program.
- Naučí se číst kód.
- Jednotlivé bloky kódu jsou pro přehlednost barevně ohraničeny.
- Přirozeně se seznámí s tečkovou notací.
- Jsou vyžadována povinná i obecně uznávaná pravidla pojmenování metod.
- Metody se při vykonávání postupně rozbalují, jsou součástí kódu, řízení programu nikam neskáče, což je pro studenty názornější a přehlednější.

- Zvlášť při užití rekurze, hlavně nechtěné, ať už přímé či nepřímé, je jasně viditelné vnořování.
- Přirozené použití volby a opakování – žádné umělé početní příklady vyžadující podmínky a cykly. Učí se rozeznávat možnosti opakování.
- Osvojí si použití negace podmínek.
- Při návrhu a programování algoritmů jsou studenti v úlohách přirozeně vedeni k vytváření metod – strukturování programu, nepíší vše do jedné metody.
- Přirozený tlak na obecnost algoritmu použitím náhody – umístění a orientace robota, velikosti plochy.
- Cyklus `for` je pro jednoduší osvojení principů nahrazen cyklem `opakuj`.
- Názornější snadnější intuitivní pochopení a užití pojmů třída, instance. Díky vytváření metod robota (naučí se jeden, umí všichni) a jejich užití v obecných třídách.
- Ukázka jednoduché dědičnosti.
- Až na OOP sada navržených úloh pokrývá RVP [6] stanovené cíle.
- Možnost přirozeně navázat dalšími objektově orientovanými kurzy programování, např. [5].

Hlavní nevýhody:

- Studenti nepíší kód – nepamatují si a nezažijí si syntaxi. Např. po přechodu do běžných vývojových prostředí neví kde a jaké musí být závorky.
- Nenaučí se tedy odstraňovat formální chyby v syntaxi - dekódovat chybová hlášení vývojového prostředí a překladače.
- Protože studenti nepíší kód, je pro ně mnohdy přirozeně jednoduší něco rychle tzv. naklikat než používat cykly a dílčí metody, musí k tomu být vyučujícím neustále vedeni.
- Při běžným programování neprobíhá automatický refaktoring při přejmenování metody.
- Zjednodušení cyklu for nahrazením opakuj je pouze počáteční výhoda.
- Metody nemají žádné parametry (Proč jsou za vším prázdné závorky?).
- Nelze vkládat a využívat vzorce.
- Nelze najednou používat vícero podmínek a vázat je pomocí logických spojek.
- Nelze vytvořit novou metodu z části kódu jiné metody.
- Nepraktické, omezené úlohy (studenti si nemohou cokoliv naprogramovat).
- Nepoužívají žádnou dokumentaci, nápovědu (nepotřebují ji).
- Zamlčujeme, neobjasňujeme vytváření instancí.
- V rámci požadovaných cílů OOP je nutné další pokračování v jiných vývojových prostředích.

2.2.2 Naplňování stanovených cílů

Navržené úlohy a práce s aplikací nemohou zcela naplnit stanovené cíle, slouží pouze pro úvod do této oblasti. Ale přesto sledují a umožňují naplnit řadu stanovených cílů. V metodických pokynech jsou pro přehlednost cíle rozděleny do čtyř oblastí – programování v této aplikaci, jazyk JAVA, algoritmizace úloh a OOP.

1. Studenti se musí postupně naučit ovládat prostředí této aplikace.
2. Osvojí si základní pravidla syntaxe a programovací konstrukce jazyka JAVA.
3. Studenti jsou především vedeni k osvojení si principů algoritmizace úloh.
4. V rámci OOP stanovené cíle slouží pouze jako úvodní seznámení s problematikou.

Při vhodném začlenění navržených úloh do výuky přispívají také k rozvoji klíčových kompetencí studentů, zejména kompetencí k řešení problémů. U ostatních kompetencí záleží na formě zvolené učitelem. Je vhodné do výuky zařazovat prezentace řešení a argumentace jeho vhodnosti, nechávat studenta nastínit problémy vzniklé při řešení, používat různé formy skupinového hledání možných řešení (exploze nápadů bez kritiky až poté společné hledání jejich kladů, záporů a případných doladění), případně využívat programování ve dvojicích.

Dílčí cíle jsou podrobně popsány v jednotlivých úlohách. Stručný souhrn cílů, které v jednotlivých oblastech sledujeme jsou:

Jazyk Java

O samotné JAVĚ se toho studenti, alepoň při řešení navržených úloh, mnoho nedozví. Z jazyka JAVA si kromě základních pravidel syntaxe postupně osvojí následující programové konstrukce.

- Cyklus `for` je pro jednoduší osvojení principů nahrazen cyklem `opakuj`.
  - První jazyková programová konstrukce – jasně definovaná, nic složitého, žádný jazykový problém.
  - Přirozené užití.
  - Jediný český výraz (kromě ano-ne). Při přechodu na skutečný cyklus se to studentům nebude plést, naopak mohou ocenit jeho bohatost.
- Cyklus `while`, včetně možné negace podmínky opakování.
- Blok podmíněných instrukcí `if-else`.

Algoritmizace

V průběhu vytvořeného kurzu si student na příkladech osvojí co to je, jaké má vlastnosti a jak se navrhuje algoritmus. Cíle v jednotlivých úlohách jsou voleny dle [8], kde autor uvádí: algoritmus je základní matematický pojem. To znamená, že jej nelze definovat – musíme se uchýlit k opisu, podobně jako u dalších elementárních pojmů, jakými jsou např. bod nebo množina. Algoritmus je v podstatě návod, jak provést určitou činnost. Ovšem ne každý návod představuje algoritmus. Jako algoritmus budeme označovat návod, který má následující vlastnosti:

1. Je **elementární**. To znamená , že se skládá z konečného počtu jednoduchých, snadno realizovatelných činností, které budeme označovat jako kroky.

2. Je **determinovaný**. Po každém kroku lze určit, zda popisovaný proces skončil, a pokud neskončil, kterým krokem má pokračovat.
3. Je **konečný**. Počet opakování jednotlivých kroků algoritmu je vždy konečný. Algoritmus tedy musí skončit po konečném počtu kroků .
4. Je **rezultativní**. Vede ke správnému výsledku.
5. Je **hromadný**. To znamená , že algoritmus můžeme použít k řešení celé (velké) skupiny podobných úloh.

V souvislosti s algoritmy se pro označení objektu, který bude provádět popisovanou činnost, používá termín procesor (může to být stroj nebo i člověk). Je jasné, že při formulaci algoritmu musíme procesor znát – musíme vědět, jak vypadají elementární kroky, které může návod obsahovat.

V našem případě se bude jednat o elementární kroky robota.

Autor v [8] dále uvádí: jak dospět k formulaci (algoritmu), který bude splňovat výše uvedené podmínky? Samozřejmě nejprve musíme daný problém umět vyřešit. Jestliže již řešení problému známe, potřebujeme je zapsat jako algoritmus, tj. rozložit na elementární kroky. Přitom postupujeme obvykle tak, že postup řešení rozkládáme na jednodušší operace, až dospějeme k elementárním krokům. Tento postup návrhu algoritmu se obvykle označuje jako metoda **shora dolů**.

Autor v [8] dále pokračuje: kromě metody shora dolů se občas setkáme i s postupem označovaným jako **návrh zdola nahoru**. Při postupu zdola si postupně z elementárních kroků vytváříme prostředky, které nakonec umožní zvládnout požadovaný problém. Obvykle se kombinuje postup shora dolů s postupem zdola nahoru.

OOP

Studenti ihned z počátku pracují s objektem (robot), který má vlastnosti a metody. Studenti si dále osvojí pojem třída a její instance, ale už jim zamlčíme nutnost instance vytvářet, v každé úloze jsou prostě k dispozici. Autor [8] uvádí tři pilíře objetově orientovaného programování: zapouzdření, dědičnost a polymorfizmus. V rámci navržených úloh se studenti mohou seznámit (spíše si jen uvědomit) se zapouzdřením a jednoduchou dědičností. Zapouzdřeny jsou vlastnosti robota (orientace, stav – výbuch, stav ano-ne), nelze je přímo měnit, můžeme volat jen metody robota. Jednoduchá dědičnost se projevuje ve vztahu tříd `Robot` a `RobotTuzka`. S tím, že metody nelze přepisovat, takže polymorfizmus se prostě nemůže projevit a studenti se s ním tady neseznámí. Stejně jako se neseznámí se skládáním a dalšími vlastnostmi OOP.

2.2.3 Kam dál?

Po úvodu do programování se mi zdá nejvhodnější pokračovat ve vývojovém prostředí BlueJ kurzem programování, který popisuje Rudolf Pecinovský ve své knize: *Myslíme objektově v jazyku Java 5.0* [5]. Ale jinak je dle mě možné pokračovat jakýmkoliv kurzem a dokonce si myslím, že to nemusí být ani v jazyce Java.

2.2.4 Poznámky ke zvolenému názvosloví

V českém názvosloví panuje zjevná nejednotnost, zapříčiněná nejspíše překladem z angličtiny – jazyka, ve kterém jsou tyto pojmy původně definovány. Dalším důvodem může být odlišnost oblastí, ve kterých pojmy v příslušném kontextu používáme.

Vycházím-li z toho, že JAVA je objektově orientovaný jazyk a studenti si mají osvojit základy objektově orientovaného programování, dospěl jsem k závěru, že je vhodné vycházet z názvosloví používaném v OOP. V UML se běžně používá „atributy a operace třídy“ (např. v [4]), většina autorů používá „atributy a metody třídy“ (např. v [1], [3], [5], [7]). Cizí slovo „atribut“ je však pro většinu studentů neznámé, na úvod ho lze tedy vyjádřit jako „vlastnost“, což je pro studenty přirozenější. Rozhodl jsem se tedy na úvod do programování používat pojmy: **vlastnosti a metody třídy**.

Avšak v úplných začátcích programování mi nepřijde vhodné používat: „robot má metody“. Osvědčilo se mi spíše: „robot má instrukce“, což studenti bez problémů přijímají. Tomuto pojmenování napomáhá i fakt, že ve specifikaci jazyka Java [2] je použito slovo „statement“, které můžeme přeložit jako instrukce či příkaz.

Autoři se běžně shodují na použití pojmu cyklus, včetně podmínky opakování u cyklu `while`. U klíčového slova `if` je ve specifikaci jazyka Java [3] uvedeno: „The if Statement“, což by se dalo přeložit jako **podmíněná instrukce** (vzhledem ke studenty používanému „instrukce robota“), a protože se nemusí jednat jen o jednu instrukci, ale blok instrukcí (v této aplikaci vždy), používám i plné **blok podmíněných instrukcí**. Často se můžeme v této souvislosti setkat s označením „podmíněný příkaz“, občas zkracovaným na „podmínka“. Toto označení by ale v této aplikaci kolidovalo s metodami, které se mohou také použít jako podmínka opakování.

V souhrnném označení již taková jednotnost není. Protože se mně to zdá v daném kontextu vhodné, používám označení **programové konstrukce** (`opakuj`, `while`, `if-else`), označení jsem převzal z [5]. Autoři uvádějí také „příkazy pro řízení chodu programu“ [7] a [1], nebo „jazykové konstrukce“ [3].

Ještě zmiňuji používání češtiny. Na střední škole je naprostá většina programů zapsána česko-anglicky. Osobně v tom nevidím problém. V této aplikaci je díky tomu úvod do problematiky pro studenty snadněji pochopitelný a intuitivnější (nejsou zahlceni informacemi). Česká slova jsou použita jen tam, kde se v syntaxi jazyka přímo nevyskytují – názvy metod, nebo jsou jednodušším ekvivalentem – cyklus `opakuj`. Ten pak při reálném programování v jazyce JAVA nahradí skutečný cyklus `for`, což by studentům nemělo činit potíže. Pro přirozenější užívání podmínek ještě v nabídce užívám místo `true-false` české `ano-ne`.

V textu jsem se tedy rozhodl používat pojem **instrukce**, což mohou být v úloze robotu do vínku dané metody, studentem vytvořené nové metody a programové konstrukce. Samozřejmě je ale možné, bez jakýchkoliv problémů, používat terminologii vlastní, například nahradit slovo instrukce slovem příkaz. V podstatě se totiž pojem instrukce používá jen v samém úvodu ve spojení s robotem, později se tohoto termínu už tolik nepoužívá a namísto toho se hovoří o metodách, cyklech a podmínkách. Studenti s tím, dle mých zkušeností, nemají problémy.

2.3 Sada úloh

2.3.1 Několik poznámek na úvod

- Text kurzu je určen pro učitele znalého probíraných principů. Není to učebnice, natož text určený k samostudiu studentů. Samozřejmě záleží na pokročilosti studenta.
- Dosažení popsaných cílů závisí na zařazení potřebné teorie učitelem a samozřejmě na stupni pokročilosti a zvídavosti studenta. Sekvence úloh je sice primárně určena pro úplné laiky, ale mohou ji procházet i pokročilejší studenti.
- Teorii o tom, co je to program, algoritmus, jeho vlastnosti a způsoby jeho návrhu vykládá učitel, kdy sám uzná za vhodné. U jednotlivých úloh je v cílech vždy určeno, co si může student uvědomit, pochopit, aplikovat.
- Pokud nejsou v cílech jednotlivých úloh zmiňované dovednosti a vědomosti nezbytné k bezprostřednímu vyřešení úkolu, jsou většinou myšleny tak, že v této úloze, případně v následujících úlohách, je možné tyto dílčí cíle naplňovat (zvláště OOP).
- V poznámkách na konci jednotlivých úloh více objasňuji smysl dané úlohy a občas přidávám své zkušenosti s použitím v praxi.
- Při tvorbě programu je vhodné vézt studenty nasledovně: definování problému → nalezení jeho řešení → návrh algoritmu → zapsani v jazyce JAVA → ladění.
- Úkoly není nutné řešit všechny a v tomto pořadí, ale doporučuji si přečíst cíle, které se mají při řešení dané úlohy naplnit, protože mnohdy se o cílech zmiňuji jen jednou, ale naplňují se průběžně i při řešení následujících úloh.
- Roboti také v navržené sekvenci úloh postupně získávají nové instrukce, stejně jako studenti postupně získávají nové informace a zkušenosti Úlohy bývají také zaměřený na právě nově přidané programové konstrukce.
- Pokud studenti vytvoří v jedné úloze nové metody, jsou nepřenositelné do dalších úloh, jsou-li potřeba, musí je studenti vytvořit znovu. Tato varianta byla přijata z důvodu možného konfliktu názvů a verzí.
  - Robot v počátečních úlohách zná jen dvě instrukce, studenti si vytvoří nové metody, které však robot už může v dalších úlohách používat jako elementární krok.
  - Studenti si mohou také v rámci uložení aktualní úlohy společně s úlohou uložit i vytvořené metody. Při načtení stejně pojmenované metody by však také mohlo dojít ke kolizi.
  - U obecných metod by se také mohlo projevit, že stejně pojmenované instance robota mohou být v různých úlohách instance různých tříd.

  Studenti samozřejmě mohou využívat vytvořené metody při řešení vícero úkolů v rámci jedné úlohy.
- Algoritmy je někdy vhodné graficky znázornit. Doporučuji využívat diagramů modelovacího jazyka UML. Zejména diagram činností. Více např. v [4].

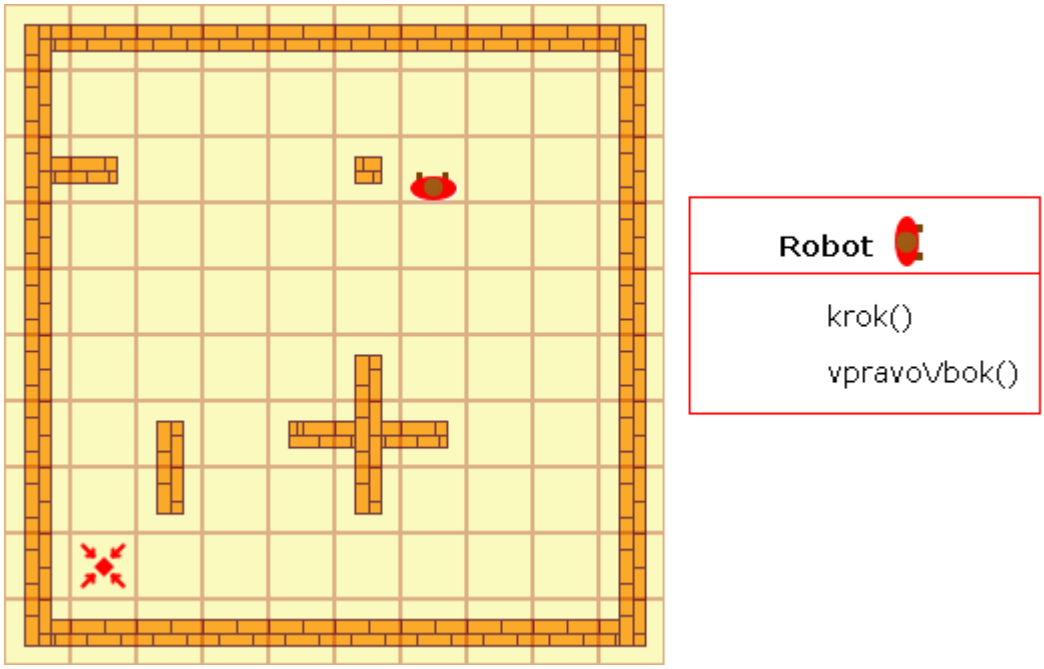
2.3.2 Úloha – první program

Úkol

Zadejte robotu takovou posloupnost instrukcí, aby po jejich vykonání stál na značce. Cestou nesmí narazit do zdi a musí vykonat všechny zadané instrukce.

Konfigurace

Úloha je v souboru `uvod1.rur`. Plocha je pevně daná a odpovídá obrázku 2.1 vlevo. Jediný robot má červenou barvu a není pojmenovaný. K dispozici jsou pouze dvě jeho instrukce (obr. 2.1 vpravo).



Obrázek 2.1: Plocha první úlohy a možné instrukce robota

Cíle

- Základní seznámení se s ovládáním prostředí
  - Student dokáže vkládat instrukce do hlavní metody, odstraňovat instrukce, měnit jejich pořadí.
  - Krokuje vytvořený program, nejsp. zjistí reakci při nárazu robota do zdi.
  - Uloží si modifikovanou úlohu.
  - Případně si prostředí přizpůsobí – velikost jednotlivých částí okna, velikost tlačítek a nastavení vlastností písma.
- Prvotní seznámení se syntaxí jazyka JAVA
  - Student si může všimnout, že názvy instrukcí – metod začínají malým písmenem a nepoužívá se interpunkce (u této úlohy pouze v názvu hlavní metody není čárka nad písmenem í),
  - název instrukce - metody `vpravoVbok()`je tvořen dvěma slovy bez mezery, pro lepší čitelnost začíná druhé slovo velkým písmenem,

  - hlavní metoda i dílčí metody mají za názvem závorky (v této aplikaci bez parametrů),
  - tělo metody je ohraničeno složenými závorkami a je nazýváno blokem instrukcí robota,
  - jednotlivé instrukce jsou odděleny středníkem,
  - pro přehlednost jsou jednotlivé instrukce na samostatném řádku a blok instrukcí je odsazen.

- OOP
  - Student si případně může uvědomit, že instrukce robota jsou graficky uspořádány v diagramu třídy a jsou to metody třídy `Robot`,
  - vlastnosti třídy (orientace robota – S, V, J, Z a jeho stav – výbuch) jsou zapouzdřeny, mohou se měnit pouze při vykonávání jednotlivých metod.

- Elementárnost, determinovanost, konečnost a rezultativnost algoritmu
  - Student si vytvoří svůj první program – poskládá smysluplně instrukce robota tak, aby robot došel na značku a program skončil (konečnost a rezultativnost).
  - Uvědomí si, že robot vždy vykonává jen jednu elementární činnost – známou instrukci (elementárnost). Instrukce vykonává postupně jednu po druhé, tedy např. nedělá krok a zároveň se neotáčí.
  - Také, že po provedení dané instrukce je vždy jednoznačně určeno, kterou instrukci má vykonat nebo vykonávání programu končí (determinovanost).
  - Nejspíše bude také program ladit než úkol splní, což se naučí spontánně (rezultativnost).

Poznámky

- Tato úloha slouží především k prvotnímu seznámení. Popsané cíle si student může uvědomit mimoděk, většina se spíše naplní při řešení dalších úloh.
- Studenti intuitivně pracují s objektem červený robot, používají jeho metody.
- Pro zjednodušení a intuitivnější seznámení s aplikací není robot pojmenovaný. Studenti vidí jednoho robota a intuitivně poskládají jeho instrukce za sebe.
- Po demonstraci principů na plátně studenti pracovali samostatně, každý u svého počítače. Jejich vytvořené programy jsme neprezentovali, studenti jen své programy spontánně srovnávali ze sousedy. Nevyskytly se tu v podstatě žádné potíže.

Navazující motivace k další úloze

Vytvořené programy (např. kód na obrázku 2.2) jsou relativně dlouhé a celé části kódu se opakují, nedá se s tím něco dělat?

```
hlavni() {
  vpravoVbok();
  vpravoVbok();
  krok();
  vpravoVbok();
  krok();
  krok();
  krok();
  krok();
  krok();
  vpravoVbok();
  vpravoVbok();
  vpravoVbok();
  krok();
  krok();
  krok();
  krok();
  krok();
}
```

Obrázek 2.2: Jedno z možných řešení první úlohy

2.3.3 Úloha – vytvoření nových metod robota

Úkol

Navrhněte jednoduší algoritmus pro řešení předchozí úlohy. Přičemž nový algoritmus se může skládat i z dalších instrukcí umožňujících robotu otočení vlevo, provést čelem vzad, krok dozadu, kroky stranou.

Poté nové instrukce robota, které jste v algoritmu skutečně použili, vytvořte jako nové metody robota (obr. 2.3 vpravo) a otestujte je voláním v hlavní metodě (otočení vlevo – `vlevoVbok()`, čelem vzad – `celemVzad()`, krok dozadu – `krokVzad()`, kroky stranou – `krokStranouVpravo()`, `krokStranouVlevo()`).

Nové instrukce použijte k sestavení programu, stejně jako v předchozím případě bude robot po vykonání všech instrukcí programu stát na značce.

Konfigurace

Plocha a elementární metody robota zůstávají stejné jako u předchozí úlohy (obr. 2.1). Jediný robot má červenou barvu a není pojmenovaný.

Cíle

- Dále přetrvávají také cíle z předchozí úlohy.
- Vytvoření a odladění nové metody robota
  - Student dokáže vytvořit novou metodu robota (příručka str. 57),
  - vhodně ji přejmenuje (uživatelská příručka str. 60),
  - vytvořenou metodu otestuje voláním v hlavní metodě (zatím nevytváříme obecné metody).
  - Vytvořenou metodu dokáže upravit i odstranit ji z nabídky.

- Syntaxe jazyka JAVA
  - Student si osvojí požadovaná pravidla pro pojmenování metody (více uživatelská příručka str. 60).
- OOP
  - Student si případně může uvědomit, že nově vytvořené metody rozšiřují třídu `Robot`.
- Návrh zdola nahoru
  - Student si při návrhu algoritmu a hledání řešení problému osvojí vytváření nových elementárních kroků (v algoritmu), čímž si postupně vytvoří prostředky, které mu nakonec umožní zvládnout požadovaný problém [8].

Poznámky

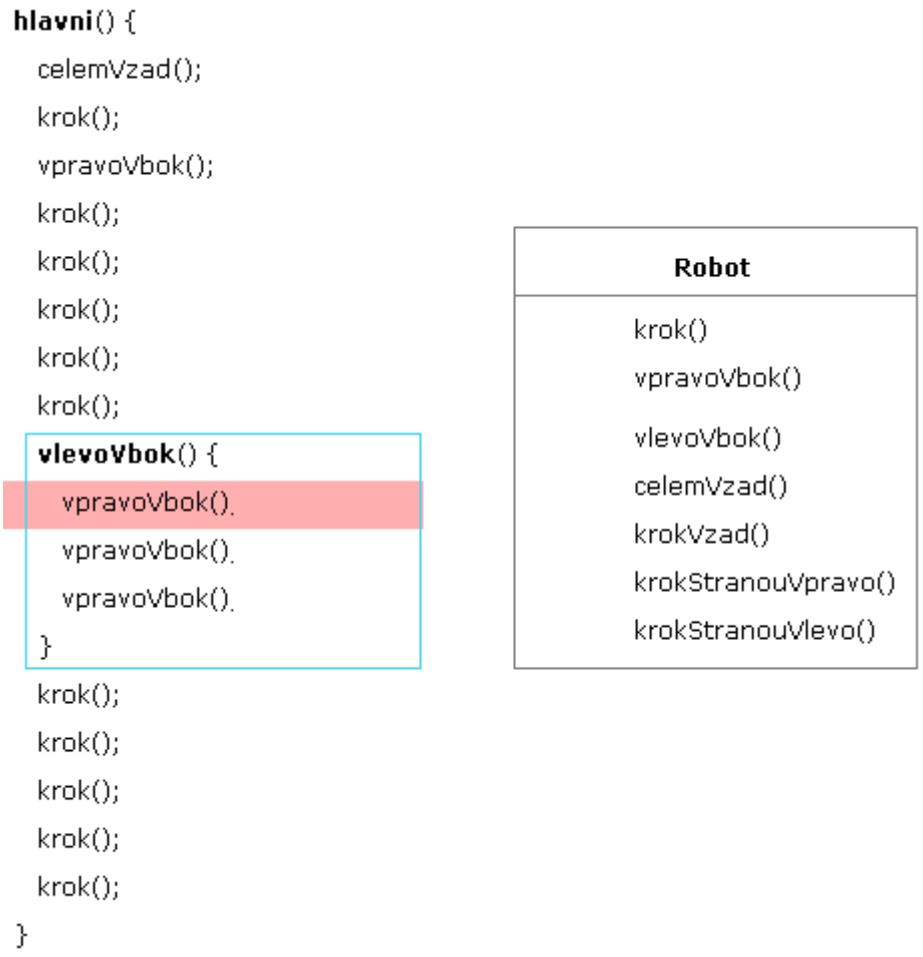
- Autor [8] uvádí, že s trochou nadsázky lze tvrdit, že při metodě zdola nahoru si vytváříme nový procesor (u nás nového robota) tím, že ho učíme nové operace (přesněji: učíme ho chápat skupiny elementárních operací jako nové elementární kroky).
- Není nutné ani vhodné vytvářet všechny možné metody, neboť se v příští úloze budou vytvářet znova, ale již s užitím možnosti opakování.
- Nově vytvořené metody se přidávají – rozšiřují třídu `Robot`. Není zde užito dědičnosti. Z pohledu aktuálních cílů to není vhodné. Prostě třídu `Robot` modifikujeme.
- Červený robot – instance třídy `Robot` ještě není pojmenována, a tak se maže rozdíl mezi metodou třídy a voláním této metody konkrétní instancí této třídy. Toto zjednodušení je provedeno cíleně kvůli snadnějšímu osvojení aktuálních cílů.
- Zatím vytváříme pouze nové metody robota, ne obecné metody. Ne, že by to nešlo, ale dané metody rozšiřují možnosti robota, z pohledu studenta se jedná o nové instrukce, které mu mohou ulehčit vytváření algoritmu a program zpřehlednit. Pokud vytvoří novou obecnou metodu, přidá se do třídy `Obecne`, což může studenty mást. Navíc to zde není ani vhodné, protože daná instance robota ještě není pojmenována, a tak není patrný rozdíl!
- Novou metodu robota nelze při upravování v její záložce přímo krokovat, nejdříve ji musíme konkrétní instancí robota zavolat v hlavní metodě. To zde není vůbec patrné, ale studenti ještě neví, že kromě hlavní metody lze krokovat i jakoukoliv obecnou metodu, takže se zmiňované okolnosti během výuky neprojevily jako problém.
- Studenti vidí, že se jim program zjednodušil, zpřehlednil, po vytvoření nových metod je programování rychlejší, nemusí si tolik rozvažovat, kolikrát mají co udělat, a dělají méně chyb.
- Vidí, že jednou vytvořená metoda se dá volat na vícero místech. Při krokování programu se při jejím vykonávání rozbaluje přímo v kódu, řízení programu tedy neskáče do metody a zpět. Jednak je to pro studeny přehlednější, ale hlavně je tak velmi názorně vidět případné rekurzivní volání.

- Studenti vidí, že se při vykonávání nové metody stejně provádějí elementární (původní) instrukce a mnohdy robot vykoná spousty instrukcí navíc (např. při opakovaném volání kroku stranou či vzad), což odpovídá současnému trendu programování. Důležitější než rychlost vykonávání programu, je přehlednost programu a efektivita při jeho vytváření, případně modifikaci.

Navazující motivace k další úloze

V metodách se nám mnohdy několikrát za sebou opakují stejné instrukce, například na obrázku 2.3 vlevo se nám v hlavní metodě několikrát po sobě opakuje instrukce `krok()` nebo v metodě `vlevoVbok()` se třikrát za sebou opakuje instrukce `vpravoVbok()`.

Což takhle stanovit kolikrát se mají instrukce opakovat a napsat je tak jen jednou?



Obrázek 2.3: Vlevo – řešení úlohy s využitím vytvoření nové metody, vpravo – třída `Robot` doplněná o všechny navrhované nové metody robota

2.3.4 Úloha – první cykly

Úkol

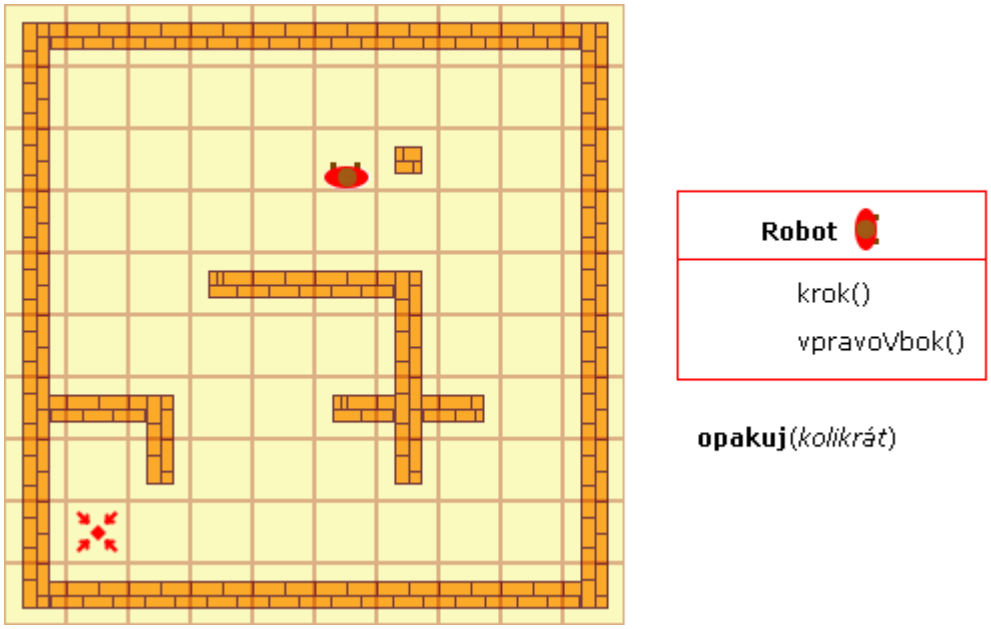
Máme k dispozici možnost nastavení počtu opakovaného provádění zadaných instrukcí – programovou konstrukci `opakuj`.

Vytvořte a otestujte metody umožňující robotu otočení vlevo, provést čelem vzad, krok dozadu, kroky stranou, tentokrát však s cíleným využitím možnosti opakování.

Upravte algoritmus pro částečně pozměněné rozmístění zdí, snažte při tom použít možnost opakování. Robot bude opět po skončení programu stát na své značce.

Konfigurace

Úloha je v souboru `2-uloha.rur`. Plocha je pevně daná a odpovídá obrázku 2.4 vlevo. Jediný robot má červenou barvu a není pojmenovaný. Na počátku jsou k dispozici pouze dvě jeho instrukce a programová konstrukce `opakuj` (obr. 2.4 vpravo).



Obrázek 2.4: Plocha úlohy a možné počáteční instrukce robota

Cíle

- Použití cyklu `opakuj`
  - Student si procvičí vytváření nových metod robota.
  - Osvojí si vkládání programové konstrukce `opakuj`, nastavení a případné změny počtu opakování (str. 54),
  - vkládání instrukce do bloku a za blok cyklu.
  - Uvidí, že pro přehlednost jsou jednotlivé instrukce na samostatném řádku a blok instrukcí je odsazen a barevně ohraničen.
  - Případně si v prostředí vyzkouší vytváření vnořených cyklů.

- Jazyk JAVA
  - Student se dozví, že programové konstrukce umožňující řízené opakování instrukcí nazýváme cykly.
  - Nastaví parametr v závorce – počet opakování.
  - Vloží opakované instrukce do těla cyklu.
  - Uvidí, že se jedná o blok instrukcí ohraničený složenými závorkami.
- Algoritmizace
  - Student si procvičí návrh algoritmu zdola nahoru.
  - Navrhne, vytvoří a otestuje nové metody robota s použitím cyklu `opakuj`.
  - Intuitivně zároveň při návrhu algoritmu použije kombinaci návrhu shora dolů.

Poznámky

- Cyklus `opakuj` je jednodušší náhražkou skutečného cyklu jazyka Java `for`. Také umožňuje pevně stanovený počet opakování vložených instrukcí, ale bez použití proměnných a má pro začínající studenty mnohem jednodušší a srozumitelnější zápis. Když se studenti naučí používat cykly, ocení možnosti cyklu `for` a bez problémů na něj v reálném programování přejdou. Neplete se jim to, protože cyklus `opakuj` je česky a je to také v této aplikaci jediné významné nedodržení syntaxe programovacího jazyka JAVA.
- Hodinu je vhodné cíleně zaměřit na použití cyklu opakuj, jinak je pro studenty jednoduší 5x na jednom místě kliknout myší, než nastavovat počet opakování.
- Studenti u těchto jednoduchých úloh plně ocenili možnost opakování, když kód ještě psali na papír.
- Cíleně vytváříme jednoduché cykly. Vnímavější studenti sami objeví možnost vytváření vnořených cyklů. Zjistil jsem, že je vhodné seznámit ostatní studenty s jejich objevem, až ve chvíli, kdy si sami trochu zažijí vytváření vlastních jednoduchých cyklů. Teprve potom to ocení, jinak to některé z nich, pro relativní složitost, odrazuje. Pokročilejší studenti si jen ověří, že to v této aplikaci jde vytvářet vnořené cykly. Pokud na to nikdo sám nepřijde, zvažte, zda-li je už v této chvíli vhodné studenty seznámit s možností vnořených cyklů. Jednu z možností řešení při využití vnořených cyklů zobrazuje obrázek 2.5.
- Student si intuitivně rozdělí cestu na několik úseků, ve kterých aplikuje opakování již připravených metod, takže intuitivně zároveň při návrhu algoritmu použije kombinaci návrhu shora dolů a zdola nahoru. Na závěr je možno tuto skutečnost studentům sdělit.
- Po samostatném snažení studentů doporučuji nechat postupně studenty vytvářet metody na dotykové tabuli a pak společně modifikovat jejich různá řešení výsledného programu.

Obrázek 2.5: Jedna z možností řešení s použitím vnořených cyklů

2.3.5 Úloha – více robotů

Úkol

Roboti mají dojít na své značky, ale nejdříve vytvořte již známé metody robota `celemVzad()`, `krokStranouVpravo()` a `krokStranouVlevo()`).

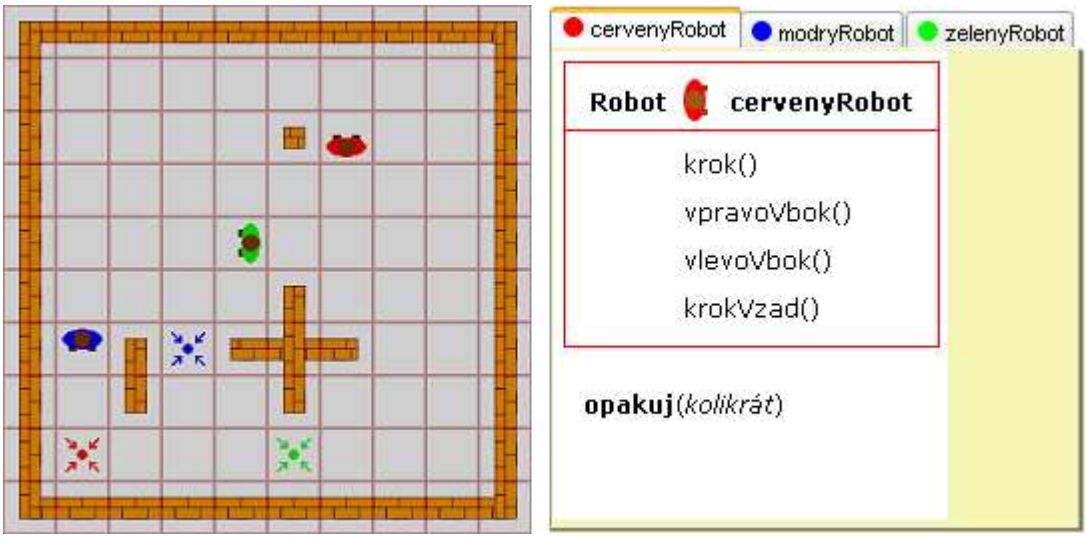
Konfigurace

Úloha je v souboru `3-uloha.rur`. Plocha je pevně daná a odpovídá obrázku 2.6 vlevo. Jsou na ní tři roboti (tři instance třídy `Robot`) pojmenovaný `cervenyRobot`, `modryRobot` a `zelenyRobot`. Na obrázku 2.4 vpravo je výřez z okna aplikace s možnými instrukcemi.

Cíle

- Prostředí
  - Student si procvičí vytváření nových metod robota.
  - Dokáže použít instrukce jednotlivých robotů.

- OOP – rozdíl mezi třídou a její instancí
  - Student zjistí, že vytvořené instance mohou ihned používat všichni roboti (všechny instance), tudíž vytváří metody třídy `Robot`.
  - Při vytváření metody robota ji nemůže přímo krokovat, není určeno, kdo ji bude vykonávat (musí ji zavolat konkrétní instance).
- Jazyk JAVA
  - Student při použití instrukcí v hlavní metodě objeví tzv. tečkovou notaci. Konkrétní instance (konkrétní robot) volá svou metodu.
- Algoritmizace – návrh shora dolů
  - Student nejspíš při vytváření algoritmu trasování jednotlivých robotů na značky intuitivně rozdělí problém na více částí, neboť se mu roboti nesmí srazit.



Obrázek 2.6: Plocha úlohy a možné počáteční instrukce robotů

Poznámky

- Cílem je především co nejintuitivnější ujasnění pojmů třída a její instance.
- Je vhodné studenty při vytváření algoritmu navést na vědomé rozdělení problému na dílčí segmenty. Nejprve půjde ten a ten robot, pak druhý.
- Případně je již možné vytvářet obecné metody.

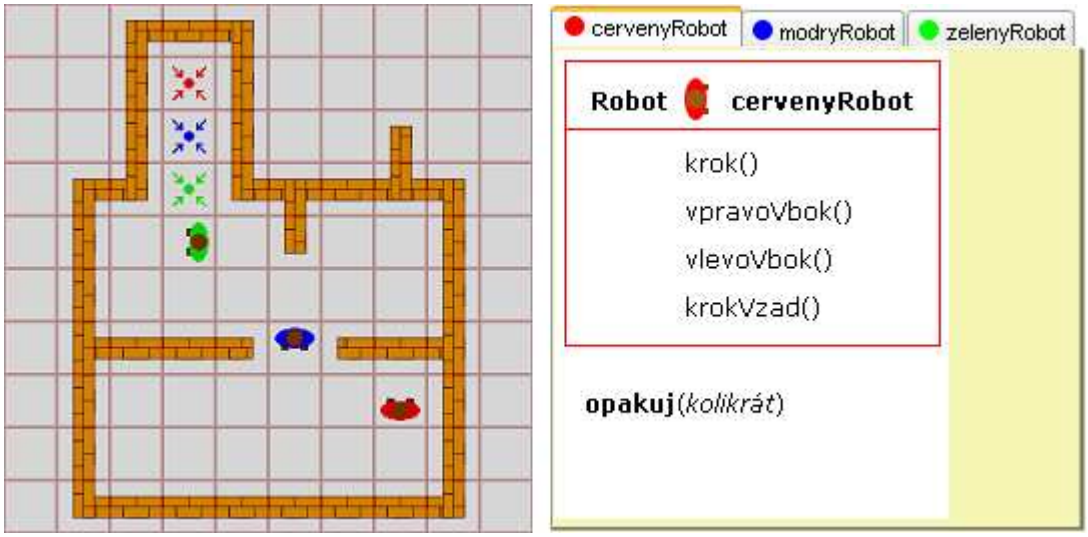
2.3.6 Úloha – vytvoření obecné metody

Úkol

Roboti mají opět dojít na své značky. Vyzkoušejte si vytvoření obecné metody, čím se liší od metody robota?

Konfigurace

Úloha je v souboru `4-uloha.rur`. Plocha je pevně daná a odpovídá obrázku 2.7 vlevo, vpravo je výřez z okna aplikace s možnými instrukcemi.



Obrázek 2.7: Plocha úlohy a možné počáteční instrukce robotů

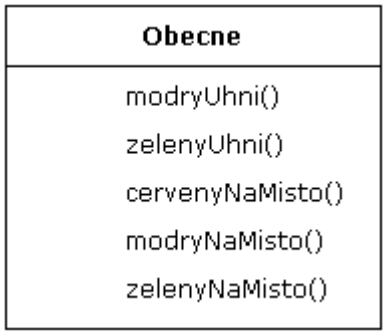
Cíle

- Prostředí
  - Student vytvoří, přejmenuje a použije obecné metody robota.
  - Vyzkouší si, že je lze přímo krokovat.
  - Dokáže je upravit a odstranit.
- OOP – třída `Obecne`
  - Student si může uvědomit, že vytvářené tzv. obecné metody jsou metody třídy `Obecne`.
- Algoritmizace – návrh shora dolů
  - Studenti cíleně vytvořením obecných metod rozdělí problém na dílčí části. Poté každou část již řeší samostatně s využitím možností opakování, čímž se úloha opět rozdělí (dekompozice úlohy).

Poznámky

- Opět je vhodné studenty při vytváření algoritmu navést na vědomé rozdělení problému, přičemž každou část algoritmu naprogramují jako obecnou metodu, kterou vhodně pojmenují tak, aby název vystihoval činnost dané metody, což vyplyne z dekompozice úlohy (například obrázek 2.8).

- Při vytváření obecných metod používáme konkrétní instance a voláme jejich metody. Z toho důvodu lze obecnou metodu přímo otestovat – krokovat, pokud to poloha robota na ploše umožní.



Obrázek 2.8: Metody třídy `Obecne`

Navazující motivace k další úloze

Při opakování víme dopředu, kolikrát se má daná sekvence provádět, ale co kdybychom to nevěděli? Třeba má robot dojít ke zdi, ale my nevíme jak je daleko. Naučíme ho to, když nám bude robot schopen odpovědět, zda-li je před ním na ploše volné políčko.

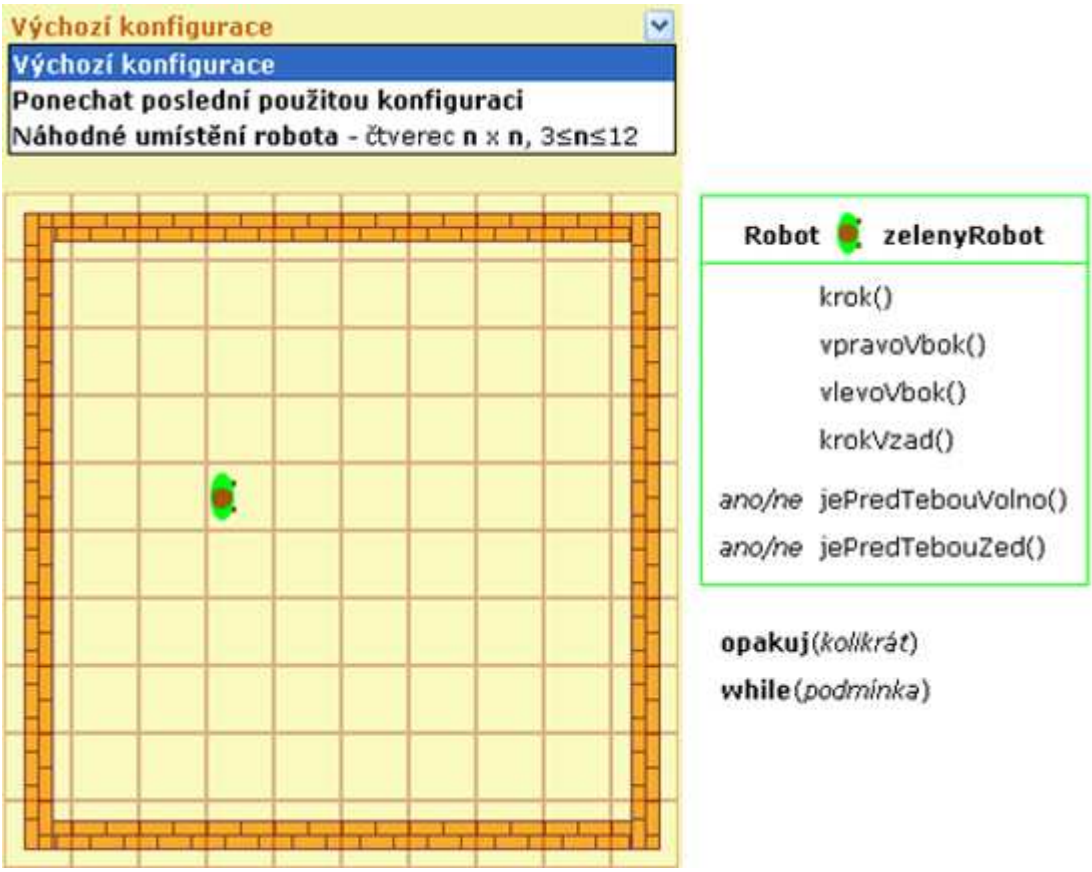
2.3.7 Úloha – cyklus while

Úkoly

1. Vyzkoušejte si jak robot reaguje při vykonávání pro vás nových instrukcí `jePredTebouZed()`a `jePredTebouVolno()`.
2. S požitím cyklu `while` vytvořte novou metodu robota `dojdiKeZdi()`, která umožní dojít robotu ke zdi, ať už stojí kdekoliv, je-li prostor plochy, kromě obvodových zdí a tohoto robota, prázdný.
3. Změňte konfiguraci a ověřte si, že metoda pracuje správně při jakémkoliv rozměru a umístění robota.
4. Čím se liší cyklus `while` od cyklu `opakuj` a v čem je stejný?

Konfigurace

Úloha je v souboru `5-uloha.rur`. Na ploše je jediný zelený robot pojmenovaný `zelenyRobot`. Možné instrukce zobrazené na obrázku 2.9 vpravo se nám rozrostly o tzv. podmínky a cyklus `while`. Možnou konfiguraci plochy volíme ze tří možností. Obrázek 2.9 vlevo zobrazuje výřez okna aplikace, nahoře je právě rozevřená nabídka možných konfigurací, pod nabídkou je zobrazena plocha a umístění robota při výchozí konfiguraci.



Obrázek 2.9: Možné konfigurace, plocha výchozí konfigurace úlohy a počáteční instrukce robota

Cíle

- Prostředí
  - Student dovede vložit cyklus `while`, nastavit podmínku opakování, včetně případné negace.
  - Případně dovede podmínku změnit.
- Jazyk JAVA
  - Student si osvojí funkčnost a použití cyklu `while`, včetně podmínek a negace.
  - Student se dozví – uvidí, jak se zobrazuje negovaná podmínka.
  - Studenti objeví v čem se cyklus `while` liší od cyklu `opakuj`.
- Algoritmizace
  - Studenti se procvičí v hledání (uvědomování si) prvků opakování, v tomto případě jen kroku.
  - Studenti vytvoří dostatečně obecný algoritmus, vhodný pro jakoukoliv dopředu neznámou vzdálenost od zdi, čímž se seznámí s další vlastnosti algoritmu – hromadostí. Hromadnost znamená [8], že algoritmus můžeme použít k řešení celé (velké) skupiny podobných úloh.

Poznámky

- Cílem je především osvojení si cyklu `while`, podmínek a negace.
- Z počátku ponecháme výchozí konfiguraci zobrazenou na obrázku 2.9 vlevo. Náhodné rozměry plochy použijeme až při testování vytvořené metody. Případně, projeví-li se nějaká chyba, si pro odladění můžeme nastavit ponechání poslední náhodně nastavené plochy.
- Při zkoumání reakcí robota při vykonávání podmínek můžeme studenty upozornit, aby si robota pomocí několika instrukcí dovedli ke zdi.
- Při objevování cyklu `while` je vhodné nechat studenty vyzkoušet si obě podmínky a jejich negace.
- Při negaci podmínky robot zobrazí odpověď na podmínku, až cyklus `while` případně reaguje na negaci.

2.3.8 Úloha – vnořené cykly

Úkol

Navrhněte a naprogramujte metodu robota, díky níž robot místnost obejde kolem dokola.

Konfigurace

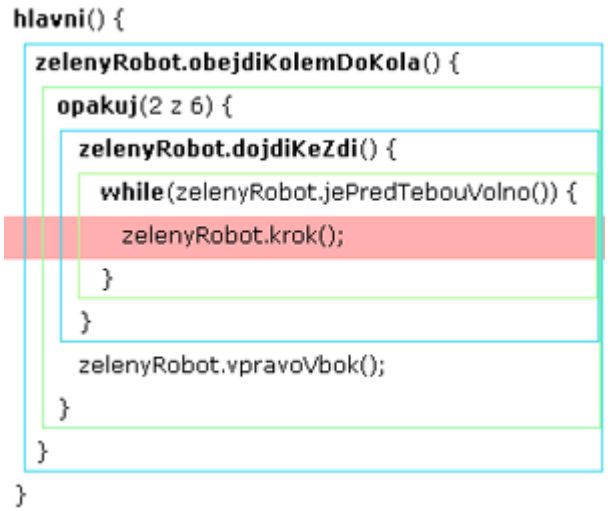
Zůstává stejná jako u předcházející úlohy (str. 24).

Cíle

- Student uvidí zobrazení vnořených cyklů.
- Algoritmizace
  - Studenti se procvičí v hledání – uvědomování si prvků opakování, v tomto případě metody umožňující robotu dojít ke zdi.
  - A zároveň při návrhu použije dekompozici úlohy.

Poznámky

- Cílem je především objevení možnosti opakování. Je vhodné nechat studenty najít vlastní řešení a až poté je případně navádět, protože někteří mohou prostě 6x za sebou napsat dojdi ke zdi a zaboč. A až potom si uvědomí, co se opakuje, nebo kolikrát se to musí opakovat. Možné řešení při krokování je vidět na obrázku 2.11.
- I když se úloha jmenuje vnořené cykly, jejich přímé užití není smyslem této úlohy. V lepším případě použijí metodu `dojdiKeZdi()` a na to, že je vlastně uvnitř jednoho cyklu druhý vnořený cyklus, je můžeme upozornit při krokování. Případně je navedeme zopakovat přímo cyklus `while`, chceme-li je tady s vnořenými cykly seznámit. Může to také ostatním na dotykové tabuli předvést student, který to sám použije.



Obrázek 2.10: Možné řešení

2.3.9 Úlohy – možnosti opakování

Úloha se skládá z několika dílčích úkolů, které slouží především k zažití hledání možností výhodných opakování při návrhu algoritmů, tedy navržení a vytvoření vhodných dostatečně obecných metod robota, jejichž opakovaným užitím vyřeší zadané úkoly. Na úkolech, kde se velikost plochy nemění, tudíž známe dopředu počet opakování, si studenti procvičí užití cyklu `opakuj`. Úkoly, kde nejsou dopředu známy rozměry zase vedou studenty k větší obecnosti (hromadnosti) navržených algoritmů a k užití cyklu `while`.

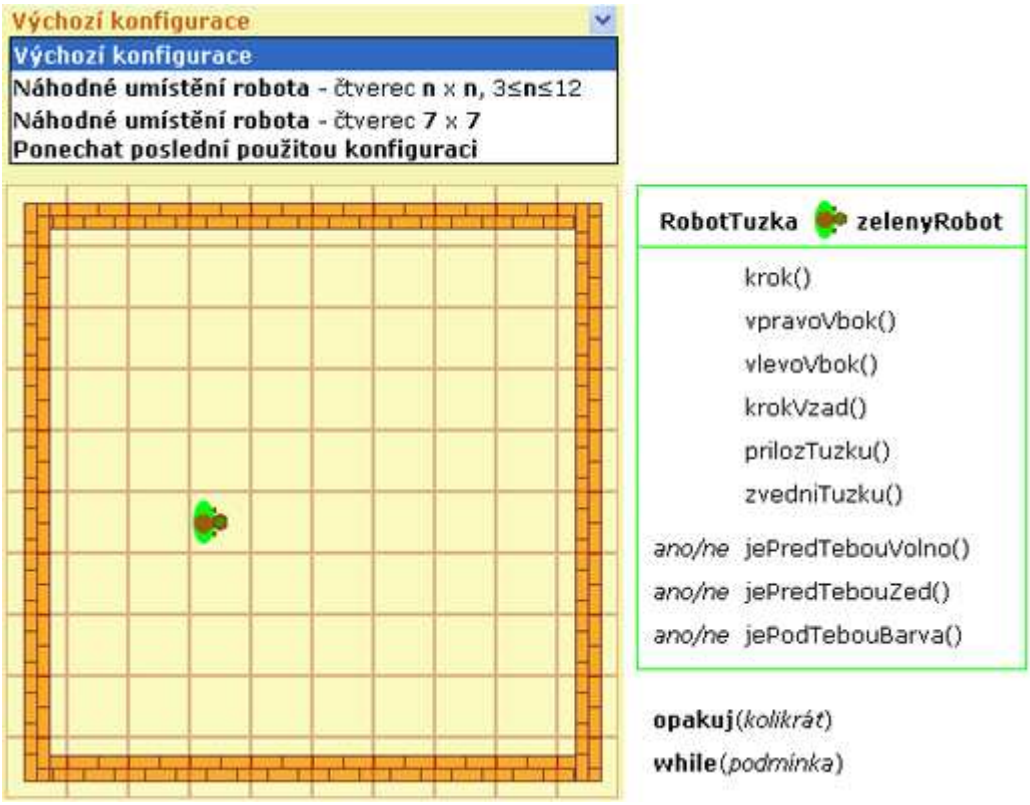
Konfigurace

Úloha je v souboru `6-uloha.rur`. Na ploše je jediný zelený robot pojmenovaný `zelenyRobot`, je instancí třídy `RobotTuzka`, která je odvozená od třídy `Robot` a má navíc metody umožňující robotu kreslit. Možné instrukce jsou zobrazené na obrázku 2.11 vpravo. Konfiguraci plochy lze volit ze čtyř možností. Na obrázku 2.11 vlevo je pod rozevřenou nabídkou možných konfigurací zobrazena plocha a umístění robota při výchozí konfiguraci.

Úkoly

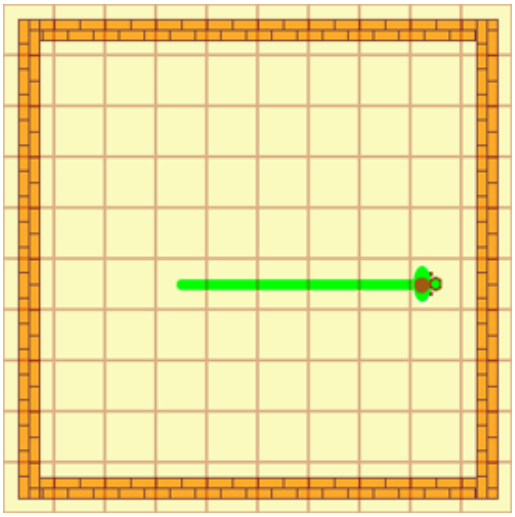
U úkolů je vždy na začátku poznamenána požadovaná konfigurace. Zprvu si algoritmus samozřejmě můžete vyzkoušet při výchozí konfiguraci. Výsledné obrázky jednotlivých úkolů odpovídající vždy výchozí konfiguraci.

1. (Výchozí konfigurace) Nejprve si vyzkoušejte, jak robot kreslí. Jak robot reaguje na instrukce `prilozTuzku()` a `zvedniTuzku()`?
2. (Náhodný čtverec) Robot nakreslí čáru z místa, kde na počátku stojí, až ke zdi. Plocha může mít jakékoliv rozměry (obr. 2.12(a)).
3. (Náhodný čtverec) Robot nakreslí obrys čtvercové místnosti neznámé velikosti. Robot startuje odkudkoliv (obr. 2.12(b)).
4. (Náhodný čtverec) Robot nakreslí body do políček z místa, kde na počátku stojí, až ke zdi (obr. 2.12(c)).

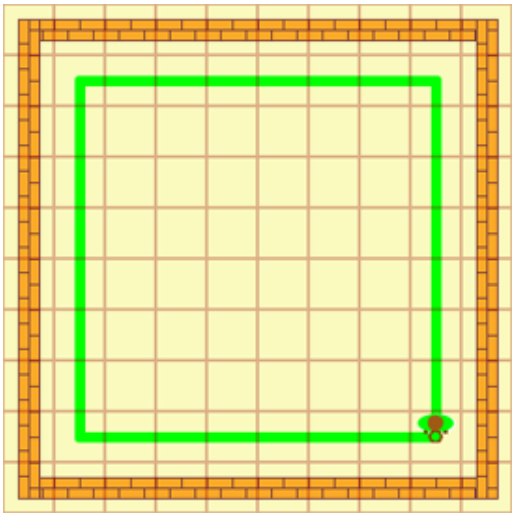


Obrázek 2.11: Výchozí plocha, nabídka konfigurací a nabídka instrukcí

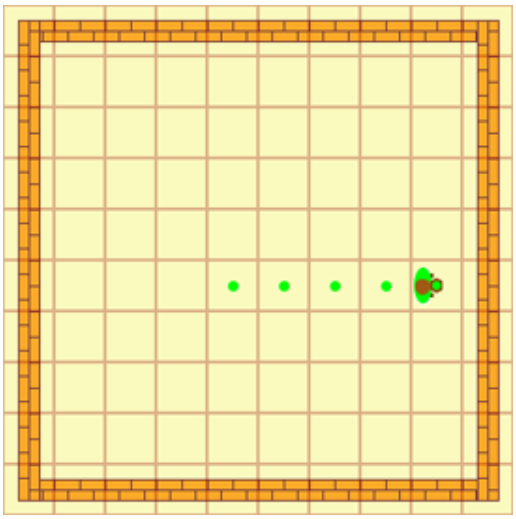
5. (Náhodný čtverec) Robot nakreslí v neznámém čtverci na úhlopříčce klikatou čáru, nesmí narazit do zdi! Obr. 2.12(d).
6. (Náhodný čtverec) Robot v neznámém čtverci vytečkuje úhlopříčky (obr. 2.12(e)).
7. (Výchozí konfigurace) Robot nakreslí cosi jako střechu, nebo spíše klobouk? Obrázek 2.12(f).
8. (Náhodný čtverec) Robot jakýmkoliv způsobem vybarví celou čtvercovou místnost (např. viz obr. 2.13(a)).
9. (Náhodný čtverec) Robot ve čtvercové místnosti vytečkuje šachovnici (obr. 2.13(b)).
10. (Čtverec 7×7) Robot se objeví kdekoliv. A u kterékoliv strany známé čtvercové místnosti nakreslí stříšku podle obrázku 2.13(c).
11. (Čtverec 7×7) Robot se objeví kdekoliv. A uvnitř známé čtvercové místnosti vytečkuje čtverec postavený na roh (obr. 2.13(d)). Jedno z možných řešení je na obrázku 2.14.



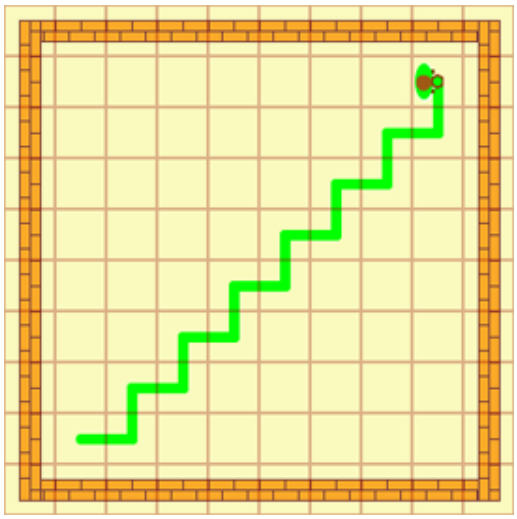
(a) 2. úkol



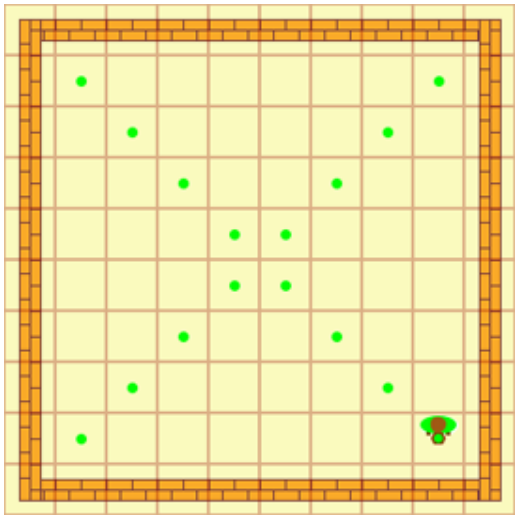
(b) 3. úkol



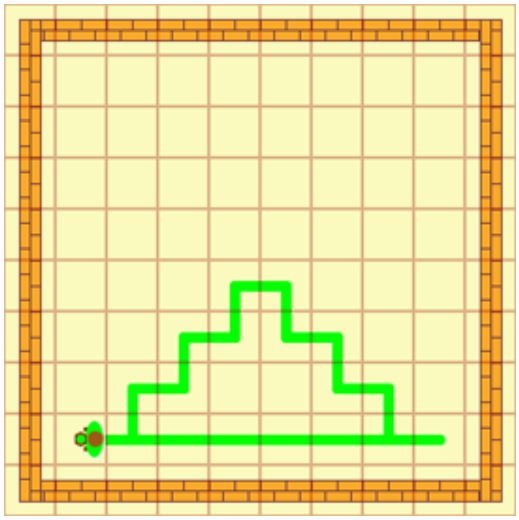
(c) 4. úkol



(d) 5. úkol

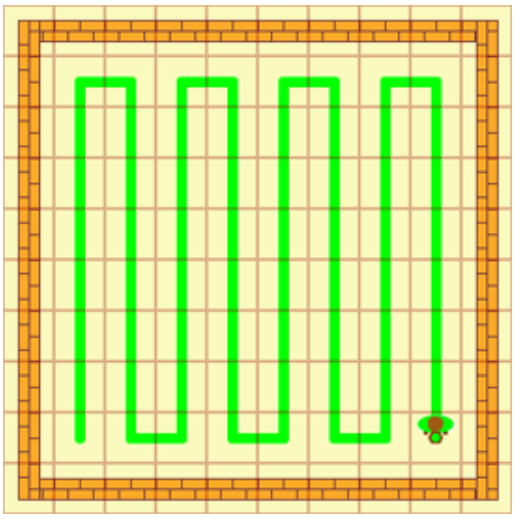


(e) 6. úkol

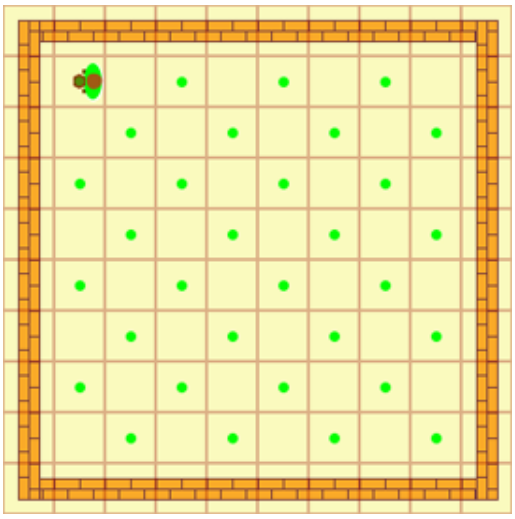


(f) 7. úkol

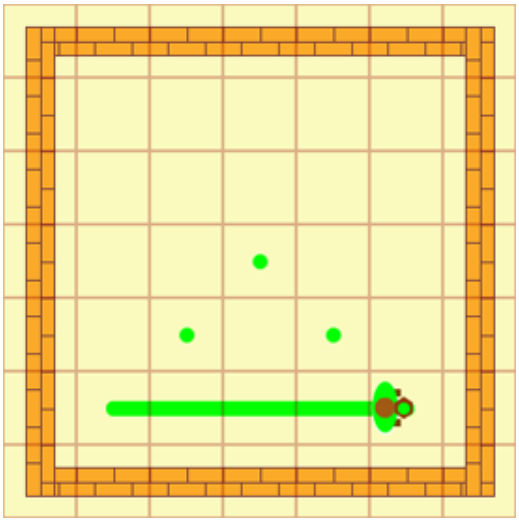
Obrázek 2.12: Výsledné plochy jednotlivých úkolů



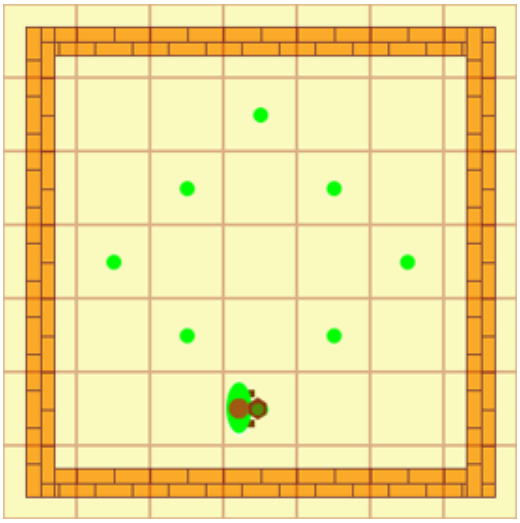
(a) 8. úkol



(b) 9. úkol

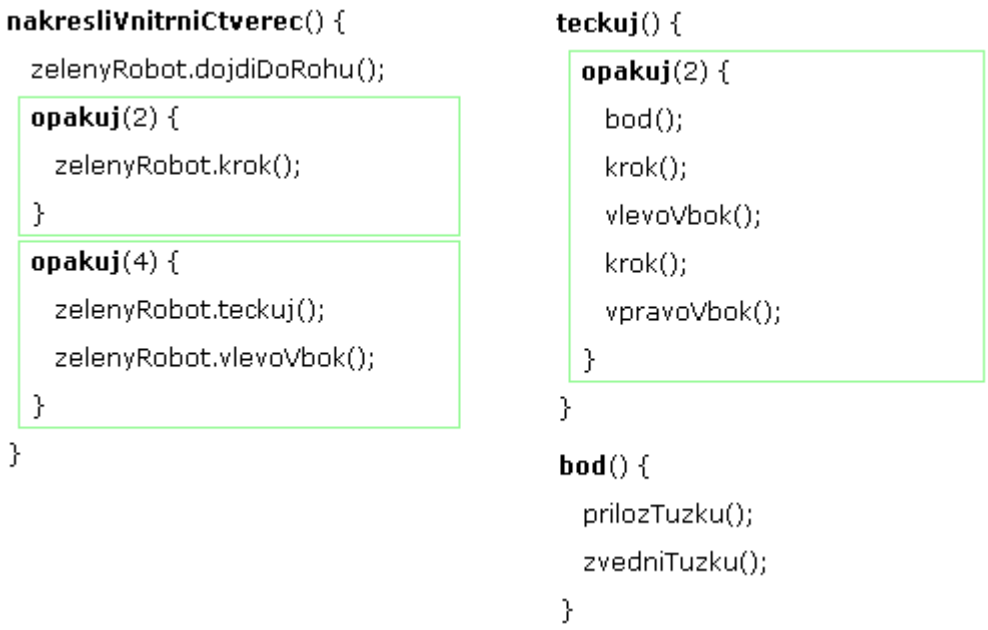


(c) 10. úkol



(d) 11. úkol

Obrázek 2.13: Výsledné plochy jednotlivých úkolů



Obrázek 2.14: Možné řešení 11. úkolu

2.3.10 Úloha – programová konstrukce if-else

Úkol

Robot má opět dojít na svou značku, ale jedna z cest je zavalená a my do okamžiku spuštění programu nevíme která. Po spuštění se náhodně místo jednoho otazníku objeví zeď.

Jak by robot zareagoval, jestliže by byly zavalené obě cesty?

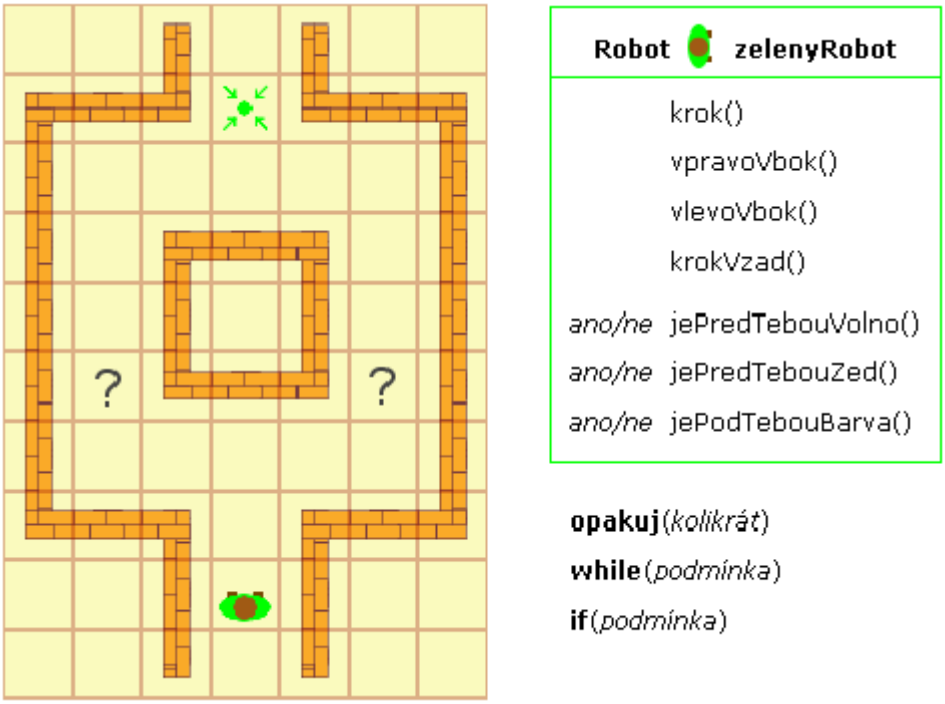
Konfigurace

Úloha je v souboru `7-uloha.rur`. Konfigurace odpovídá obrázku 2.15. Políčka s otazníky jsou tzv. náhodná políčka. Po spuštění krokování programu se v jednom z těchto políček objeví zeď, druhý otazník zmizí.

Instrukce `if` nemá v této konfiguraci implicitně blok `else`, je nutné si ho navolit při nastavování vlastností (volbě podmínky).

Cíle

- Prostředí
  - Student dovede vložit programovou konstrukci `if`, nastavit podmínku, včetně případné negace.
  - Umí využívat možností bloku `else`.
  - Dovede nastavené vlastnosti případně změnit.
- Jazyk JAVA
  - Student si osvojí funkčnost a použití programové konstrukce `if-else`.



Obrázek 2.15: Plocha úlohy a možné instrukce robota

Poznámka

- Studenti se s programovou konstrukcí `if-else` již setkali, např. fukce `když` v MS Excel nebo v hromadné korespondenci v MS Word. Její funkčnost a jednoduché využití jim tedy není cizí.

2.3.11 Úloha – kopíruj vzor

Úkol

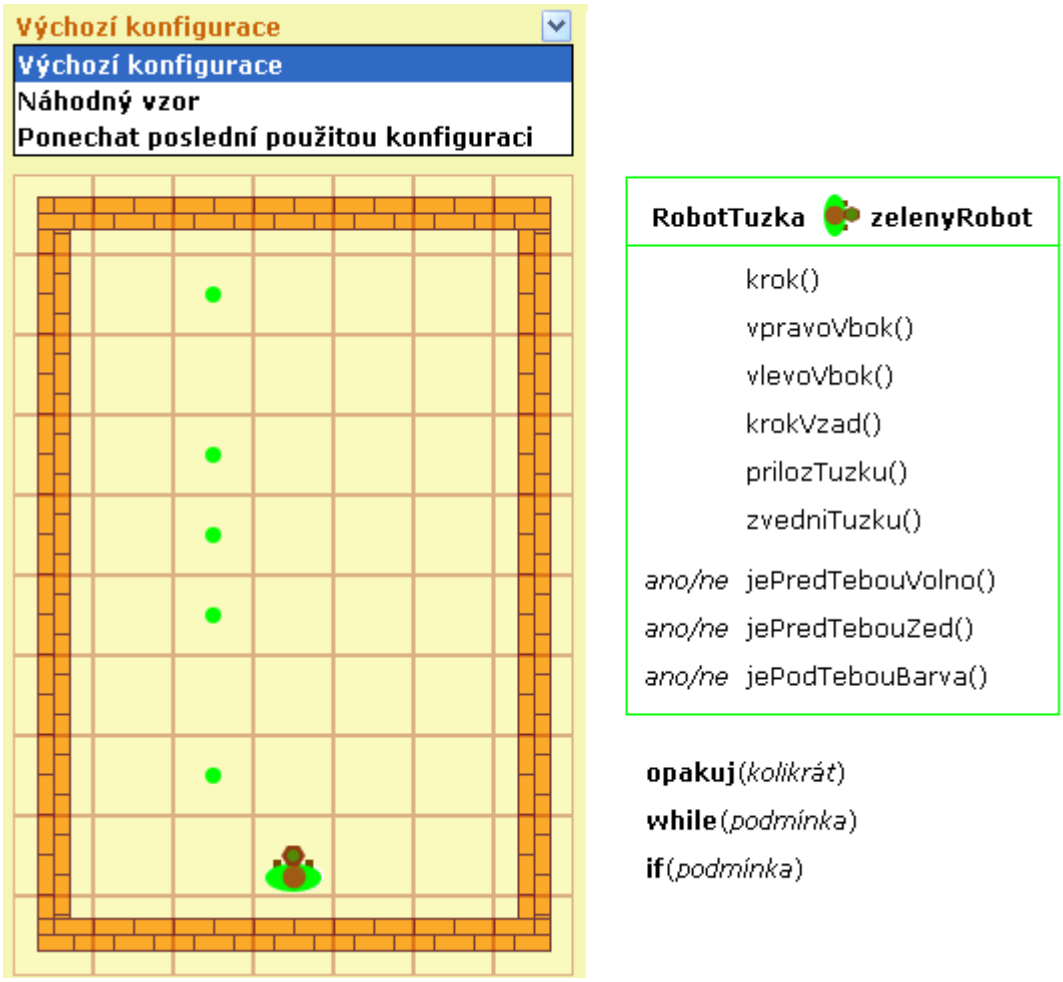
Navrhněte a naprogramujte dostatečně obecný algoritmus, aby robot ve svém sloupci nakreslil tečky podle umístění teček ve vedlejším sloupci. Přičemž vzor se při spuštění krokování nastaví náhodně.

Konfigurace

Úloha je v souboru `8-uloha.rur`. Konfigurace odpovídá obrázku 2.16.

Cíle

- Prostředí
  - Student si procvičí vkládání a nastavení vlastností programové konstrukce `if-else`.
- Algoritmizace.
  - Student si procvičí principy dekompozice úlohy s využitím opakování dílčích částí po splnění odpovídající podmínky.



Obrázek 2.16: Výchozí plocha úlohy, možné konfigrace a instrukce robota

Poznámky

Dovolím si upozornit na možná, leč očekávaná studentská pochybení. Jedná se o možné zacyklení robota při kreslení tečky a problémy s kreslením první tečky hned vedle robota nebo poslední tečky u zdi.

Navazující motivace k další úloze

Konstrukce `if-else` umožňuje vykování jedné ze dvou možností, ale co kdyby možnosti byly tři?

2.3.12 Úloha – vnořená podmínka

Úkoly:

Modrý robot má dojít na svou značku.

1. K cíli vede jedna ze tří cest, ostatní jsou zavalené a my do okamžiku spuštění programu nevíme které. Po spuštění krokování se náhodně místo dvou prvních otazníků objeví zeď a ostatní otazníky zmizí (1. konfigurace, např. obr. 2.18(a)).
2. I tentokrát k cíli vede jedna ze tří cest, ostatní jsou zavalené, ale a my do okamžiku spuštění programu nevíme kde. Po spuštění krokování se náhodně ve dvou cestách místo jednoho otazníku objeví zeď a ostatní otazníky zmizí (2. konfigurace, např. obr. 2.18(b)).
3. (∗) I tentokrát k cíli vede jedna ze tří cest, ale tato cesta je blokovaná robotem a ostatní jsou zavalené. Do okamžiku spuštění krokování programu nevíme, které cesty budou zablokované a ve které bude robot blokující cestu. Nevíme také, kde se bude tento robot nacházet a jak bude orientován. (3. konfigurace, např. obr. 2.18(c)).
4. (∗) A tentokrát bude jedna ze tří cest zablokovaná, avšak dvě zbylé blokují dva roboti. Do okamžiku spuštění krokování programu nevíme, která cesta bude zablokovaná a ve kterých se budou nacházet roboti blokující cestu. Nevíme také, kde přesně se v dané cestě tito roboti budou nacházet a jak budou orientováni. (4. konfigurace, např. obr. 2.18(d)).

Konfigurace

Úloha je v souboru `9-uloha.rur`. Výchozí plocha je na obrázku 2.17 vlevo a po spuštění krokování programu se změní dle vybrané konfigurace. Máme na výběr z pěti možností. První čtyři odpovídají jednotlivým úkolům, při volbě páté možnosti se ponechá posledně použité nastavení plochy, nebude se tedy náhodně měnit, pokud však ještě nebyla žádná možnost vygenerováva, tak se na poprvé náhodně zvolí jedna z možností prvního úkolu.

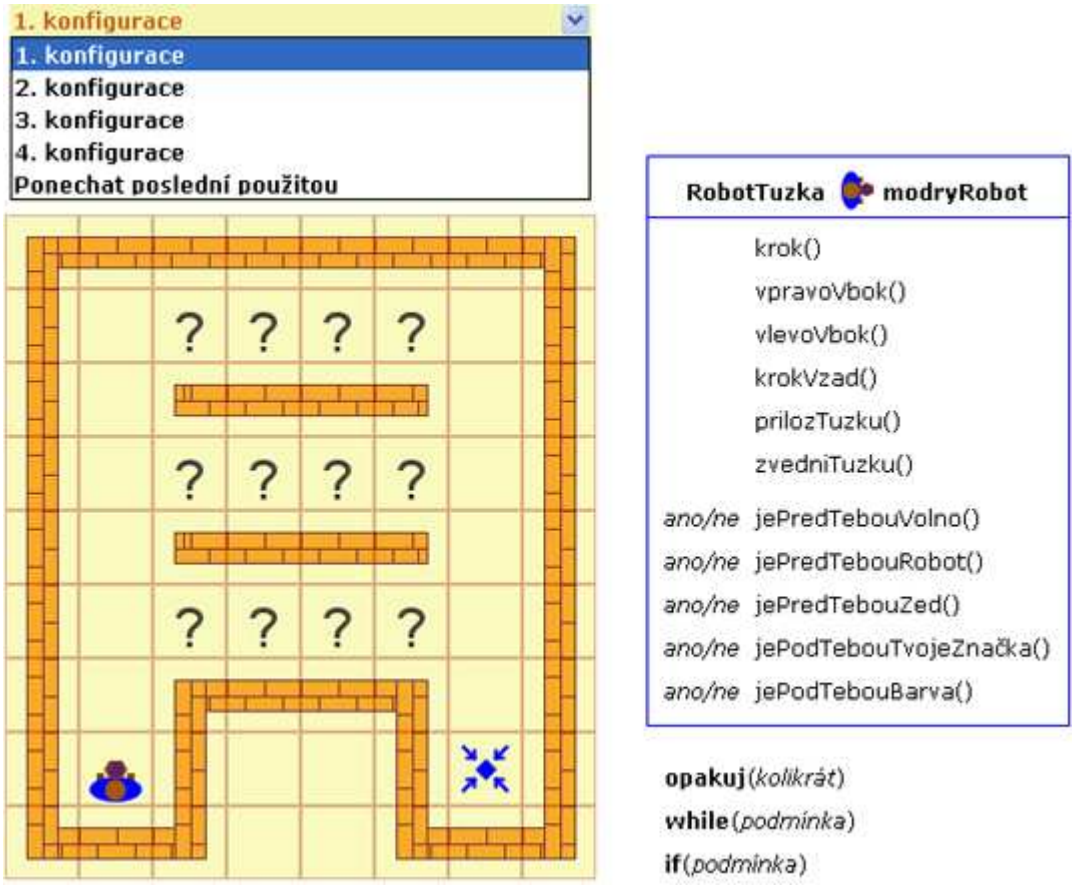
Všichni roboti jsou instancí třídy `RobotTuzka` a jejich možné instrukce v jednotlivých úkolech odpovídají instrukcím modrého robota na obrázku 2.17 vpravo.

Cíle

- Prostředí
  - Student cíleně vloží jednu konstrukci `if-else` do jiné.
- Algoritmizace.
  - Student si osvojí a procvičí principy dekompozice úlohy s využitím možnosti rozvětvení na více než dvě části postupným větvením.

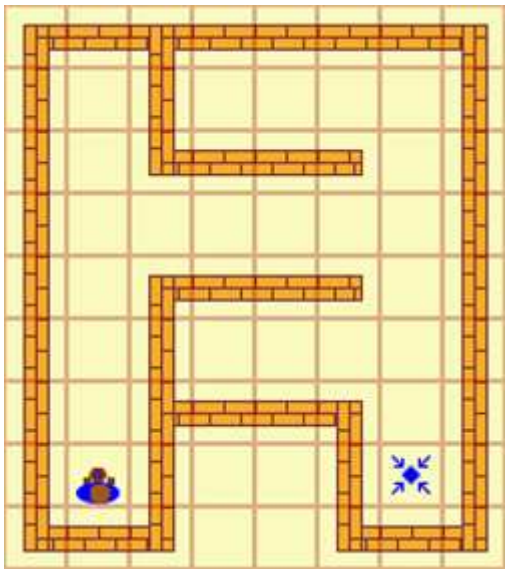
Poznámky

- První úkoly jdou velmi elegantně řešit pomocí cyklů, ale v této úloze jde především o použití vnořených podmínek.

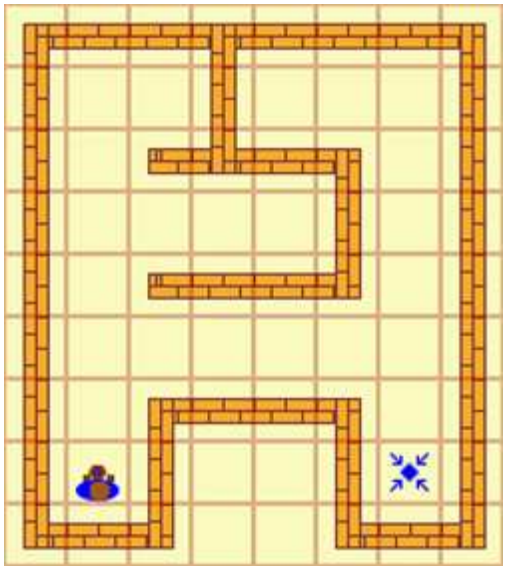


Obrázek 2.17: Výchozí plocha úlohy a nabídka instrukcí robotů

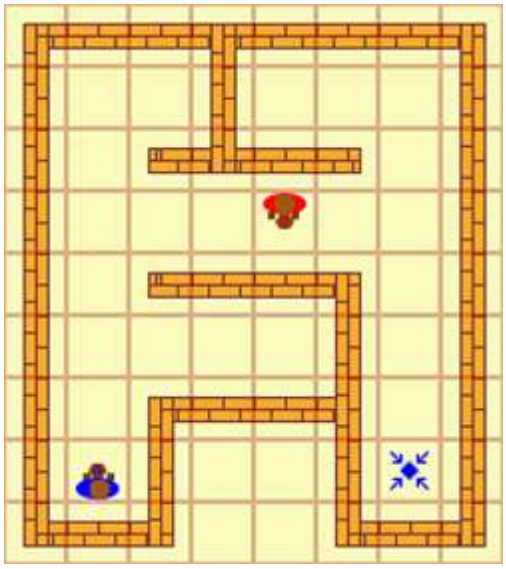
- Studenti mají k dispozici pro ně nové podmínky `jePredTebouRobot()` a `jePodTebouTvojeZnacka()`, ale při řešení je vůbec nemusí využít.
- Je to první úloha, kde může více robotů kreslit. Není to sice cílem této úlohy, ale studenti mohou případně využít i této skutečnosti.
- Při návrhu algoritmu je vhodné si nechat spuštením krokování vygenerovat konkrétní plochu a v nabídce vybrat ponechání posledně použité konfigurace, náhodná políčka tak nebudou po stopnutí krokování ihned vyplněna otazníky.
- Třetí úkol bývá již pro studenty náročný, protože blokující robot může být orientován kterýmkoliv směrem, takže neví, kde je a kam má jít. I když společně naleznete řešení problému, stejně je poměrně obtížné program odladit, zvažte tedy, jestli je tento úkol pro danou skupinu studentů vhodný.
- Poslední úkol je pak už velmi jednoduchý, studenti jen mívají problém získat nadhled potřebný k nalezení elegatního řešení tohoto úkolu. Stačí na jednoho robota pohlížet jako na zeď a v tu chvíli se nám problém převede na již vyřešený přechozí úkol.



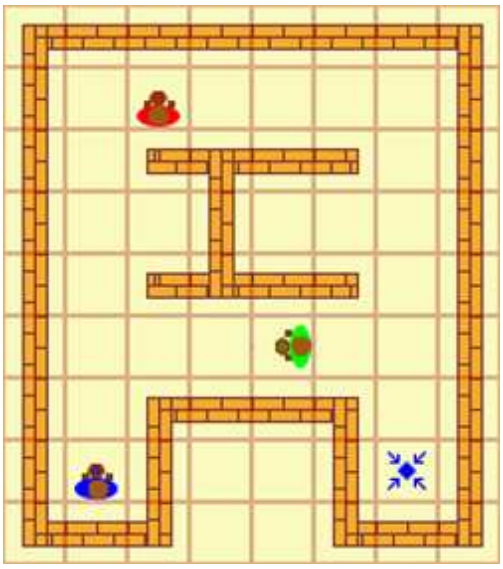
(a) 1. úkol



(b) 2. úkol



(c) 3. úkol



(d) 4. úkol

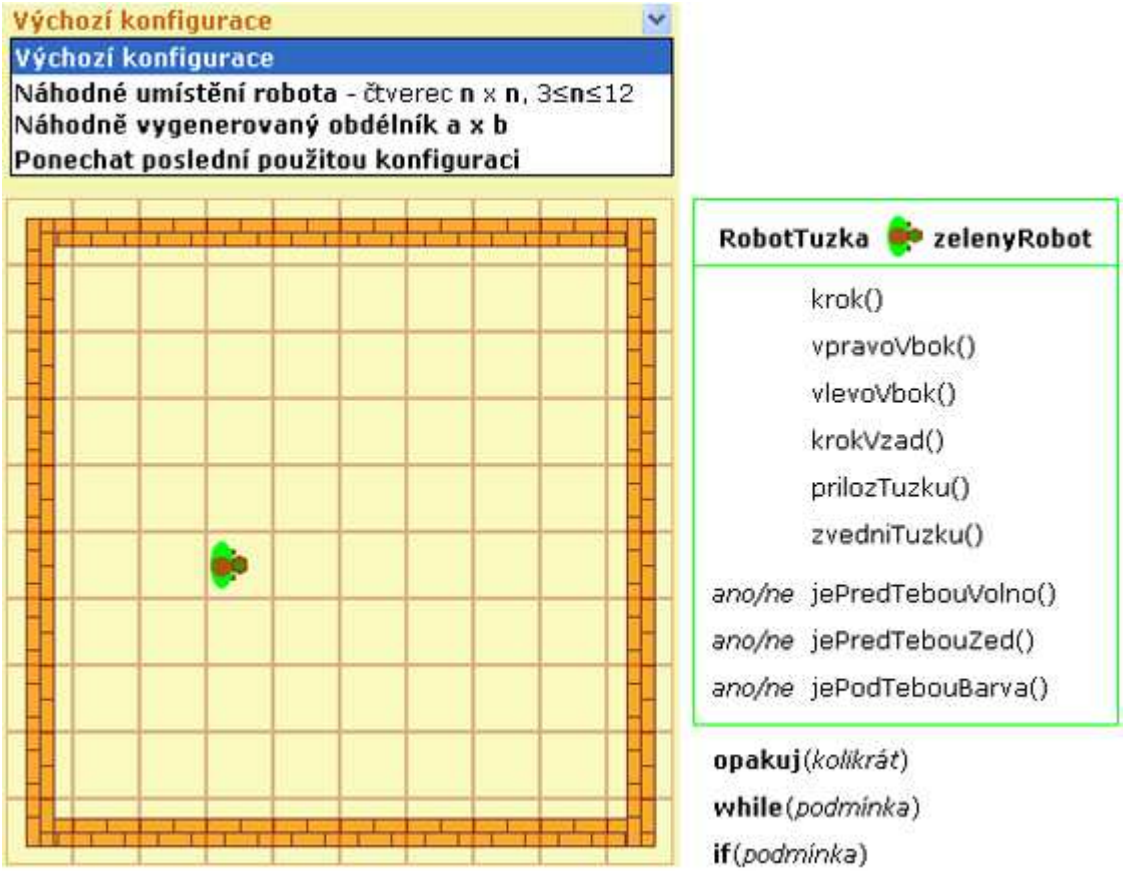
Obrázek 2.18: Příklady ploch jednotlivých úkolů po spuštění krokování

2.3.13 Úlohy – možnosti rozhodování

Úloha vychází z úkolů z kapitoly 2.11 na straně 28. Některé úkoly tedy již známe, ale tentokrát je budeme řešit s větším prvkem náhody. Úkoly slouží především k využití možnosti větvení při vytváření obecnějších (hromanějších) algoritmů.

Konfigurace

Úloha je tedy v souboru `10-uloha.rur`. Konfigurace na obrázku 2.19 odpovídá téměř konfiguraci v kapitole 2.11 na straně 28, liší se možností náhodné obdélníkové plochy a samozřejmě nabídkou podmíněné instrukce `if-else`. Ostatní je stejné, na ploše je jediný zelený robot pojmenovaný `zelenyRobot`, který je instancí třídy `RobotTuzka`.



Obrázek 2.19: Výchozí plocha úlohy, možné konfigurace a instrukce robota

Úkoly

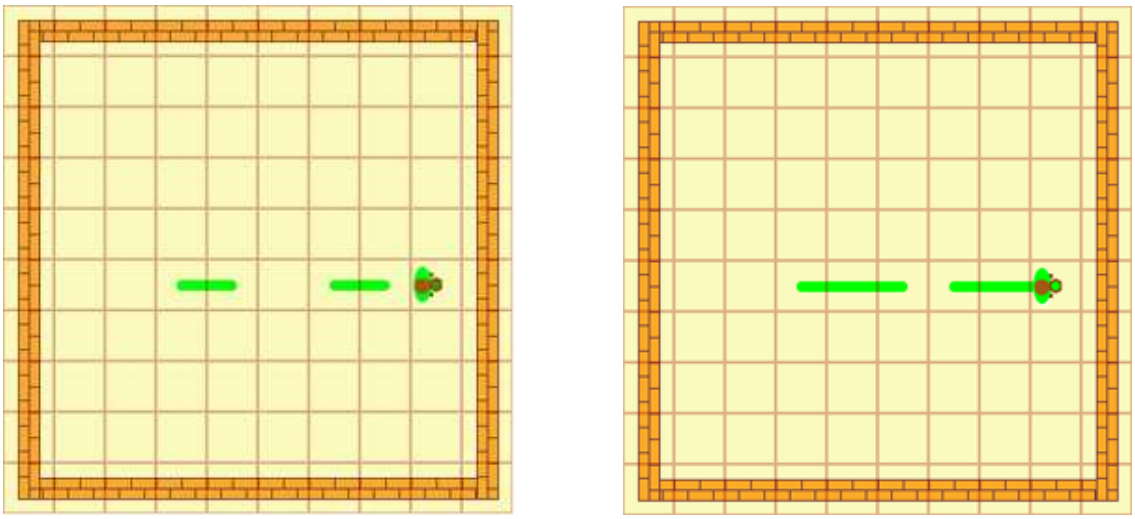
I zde je vždy na začátku poznamenána požadovaná konfigurace. Algoritmus si zprvu lze samozřejmě vyzkoušet při výchozí konfiguraci. Výsledné obrázky jednotlivých úkolů odpovídající vždy výchozí konfiguraci a pro přehlednost jsou tu znovu i poslední dva obrázky již zobrazené v kapitole 2.11.

1. (Náhodný obdelník) Robot nakreslí přerušovanou čáru z místa, kde na počátku stojí až ke zdi dle obrázku 2.20(a). Nesmí však nakreslit jen tečku. Plocha může mít jakékoliv rozměry.

2. (Náhodný obdelník) Robot nakreslí přerušovanou čáru z místa, kde na počátku stojí až ke zdi, tentokrát ale dle obrázku 2.20(b), tedy mezera je kratší a dílčí čárka delší. Ani tady nesmí před zdí nakreslit neúplnou dílčí čárku.
3. (Náhodný obdelník) Robot obtečkuje obrys místnosti neznámé velikosti. Robot startuje odkudkoliv a nesmí žádnou tečku udělat dvakrát (obr. 2.21(a)).
4. (Náhodný obdelník) Tentokrát obtečkuje obrys místnosti tak, že jedno políčko vždy vynechá (obr. 2.21(b)).
5. (Náhodný obdelník) Robot jakýmkoliv způsobem vybarví celou místnost o neznámé velikosti (např. obr. 2.21(c)).
6. (Náhodný obdelník) Robot v jakékoliv obdelníkové místnosti vytečkuje šachovnici (obr. 2.21(d)).

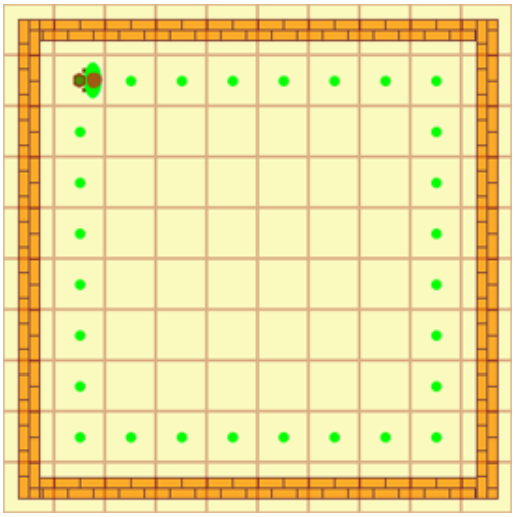
Poznámka ke 3. úkolu

Úkol lze samozřejmě řešit již použitým postupem pomocí cyklů bez užití podmínky, ale myšlenka elegantnějšího řešení, poté co robot dojde ke zdi, je: dokud nemáš pod sebou tečku, udělej tečku, (pokud je před tebou zeď, otoč se do prava) udělej krok.

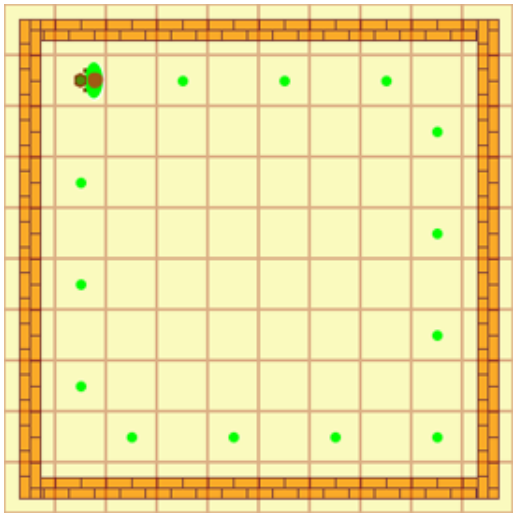


(a) 1. úkol (b) 2. úkol

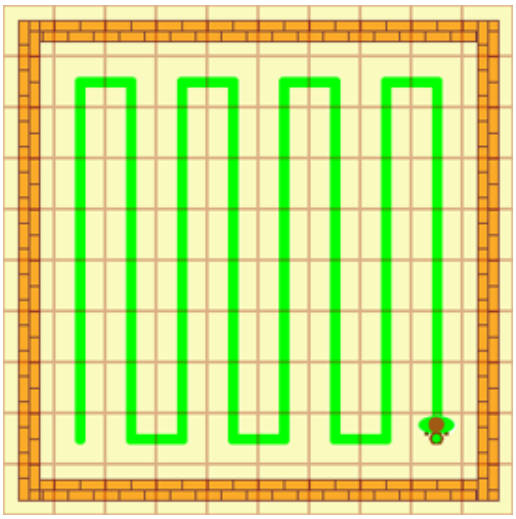
Obrázek 2.20: Výsledky jednotlivých úkolů



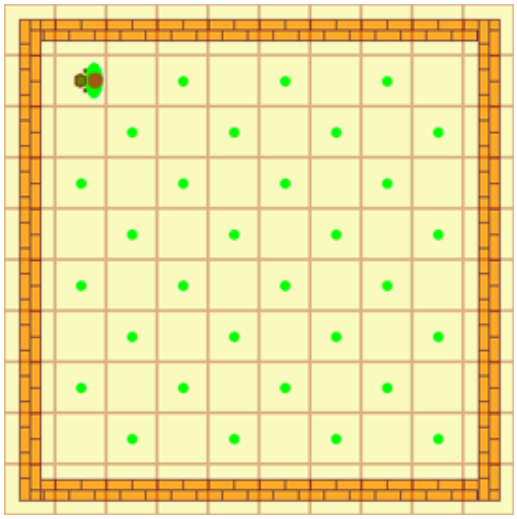
(a) 3. úkol



(b) 4. úkol



(c) 5. úkol



(d) 6. úkol

Obrázek 2.21: Výsledky jednotlivých úkolů

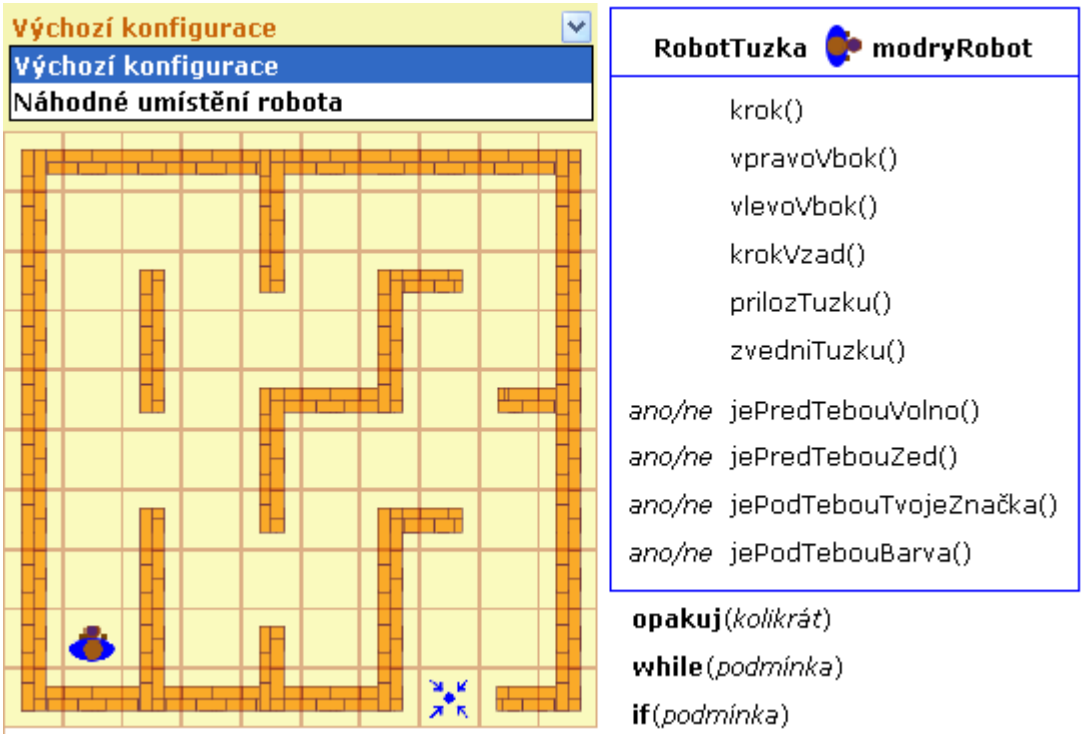
2.3.14 Úloha – vyjdi z labyrintu

Úkoly

1. Navrhněte a naprogramujte algoritmus, který umožní robotu při výchozí konfiguraci dojít v labyrintu na svou značku.
2. Navrhněte a naprogramujte dostatečně obecný algoritmus, který také umožňující robotu dojít na svou značku, ale tentokrát se může robot po spuštění krokování programu objevit na kterémkoliv volném políčku plochy a je tak možné, že by mohl chodit okolo části zdi (konfigurace – náhodné umístění robota).

Konfigurace

Úloha je v souboru `11-uloha.rur`. Konfigurace odpovídá obrázku 2.22.



Obrázek 2.22: Výchozí plocha úlohy, možné konfigurace a instrukce robota

Cíle

- Algoritmizace
  - Student si procvičí principy dekompozice úlohy při přizpůsobení známého algoritmu pro robota.
  - A zároveň je veden k návrhu obecných (hromadných) algoritmů tím, že se snaží postihnout všechny možnosti.

Poznámky

Hledáme nějaké řešení, nemusí být nejkratší. Jako řešení prvního úkolu můžete studentům nabídnout tzv. pravidlo pravé ruky (robot v podstatě kopíruje zeď po své pravé ruce), jehož implementace není náročná, ale může selhat při druhé konfiguraci, kdy robot může chodit kolem dokola části zdi. Situaci lze řešit tím, že algoritmus doplníme o jednu podmínku. Robot si na místě, kde začíná, udělá tečku a pokud ji opět najde, provede čelem vzad.

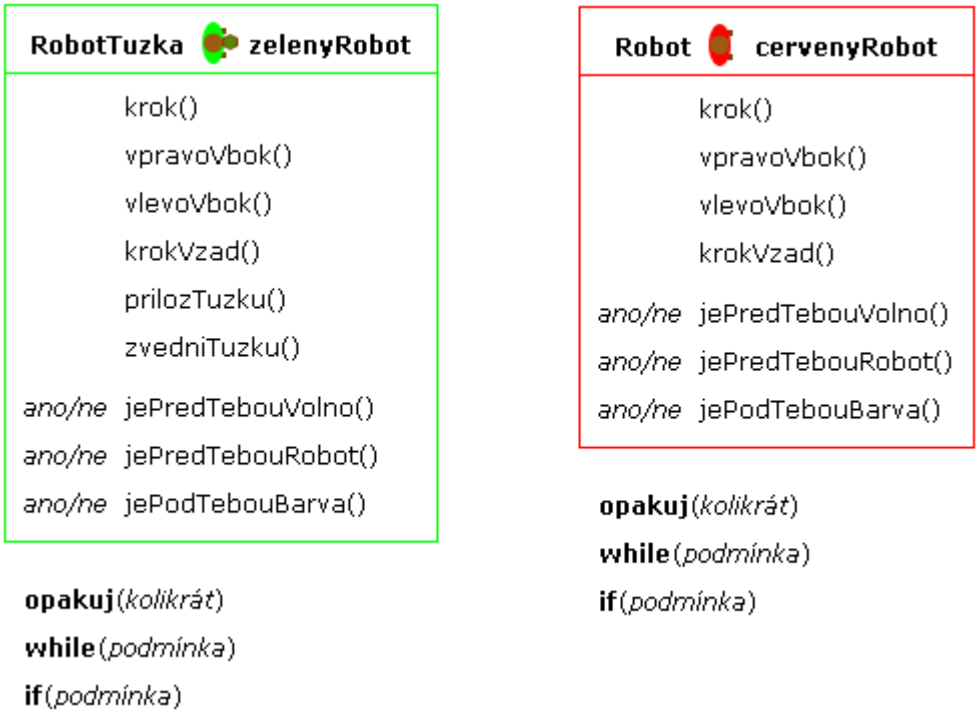
2.3.15 Úloha – následuj robota

Úkoly

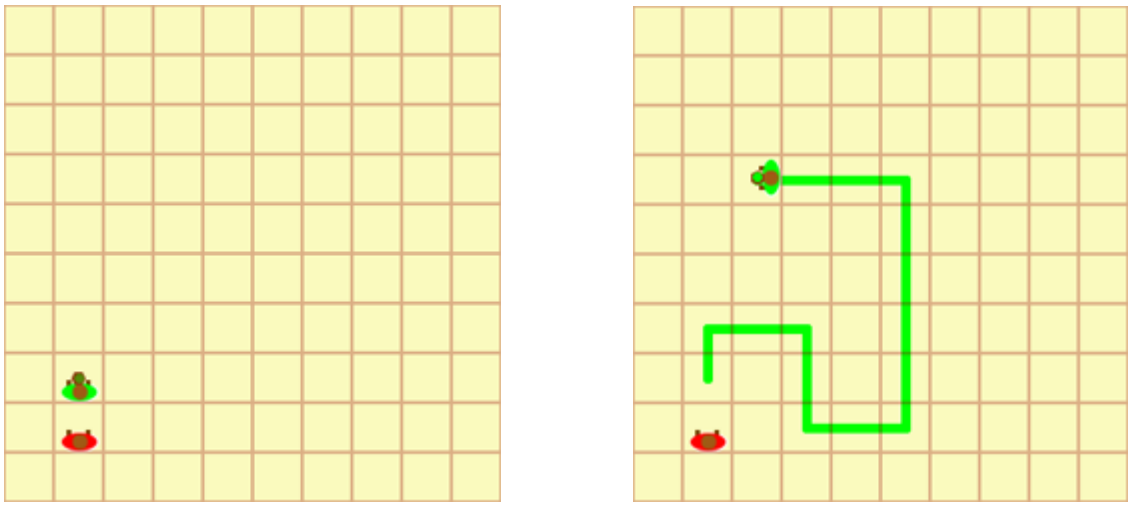
1. Nejprve naprogramujte metodu robota s tužkou, aby od druhého robota nakreslil souvislou čáru. Čára se nesmí nikde křížit a jednotlivé linie musí být od sebe a od okraje odděleny alespoň jedním políčkem (např. obr. 2.24(b)).
2. Navrhněte a naprogramujte algoritmus umožňující druhému červenému robotu dojít po čáře až k prvnímu robotu. Implemenujte ho jako novou metodu robota.
3. Na vámi vytvořeném algoritmu demonstrujte a objasněte všechny vlastnosti algoritmu.
4. Objasněte, jakou metodu návrhu jste při vytváření algoritmu použili.
5. Na konkrétních příkladech z vašeho programu objasněte funkčnost a vhodnost použití programových konstrukcí jazyka JAVA.
6. (!) Pokud jste si pro tvoření čáry robota s tužkou vytvořili jeho novou metodu zjistíte, že druhý robot (bez tužky) ji nemá v nabídce. Čím to?

Konfigurace

Úloha je v souboru `12-uloha.rur`. Plocha je vidět na obrázku 2.24(a) nemá žádné zdi a roboti se vždy zobrazí za sebou v levém dolním rohu. Jejich možné instrukce vidíte na obrázku 2.23.



Obrázek 2.23: Možné počáteční instrukce robotů



(a) Plocha po spustění

(b) Příklad možné trasy

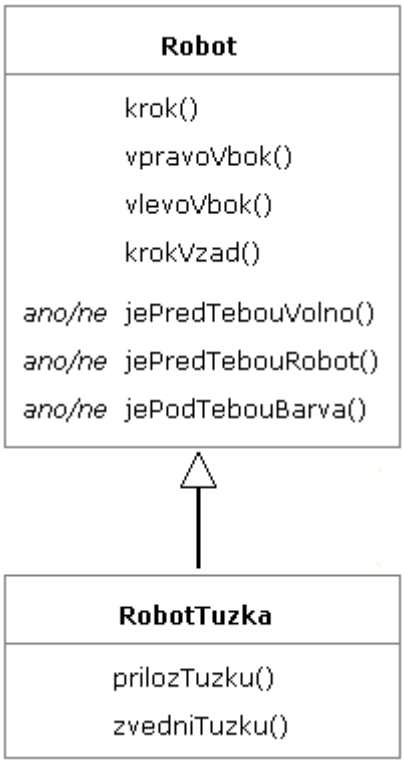
Obrázek 2.24: Úkázky ploch úlohy

Cíle

- Prostředí.
  - Student vytvoří a odladí jednoduchý program.
- Algoritmizace.
  - Student na vhodných příkladech demonstrujte a objasní všechny vlastnosti algoritmu.
  - Student je schopen sestavit algoritmus řešení konkrétní úlohy (dekompozice úlohy na jednotlivé elementárnější činnosti za použití přiměřené míry abstrakce).
- Jazyk JAVA
  - Student na příkladech demonstruje funkčnost a vhodnost použití probíraných programových konstrukcí jazyka JAVA.
- OOP
  - Student se seznámí s pojmem dědičnost. Přijme, že třída `RobotTuzka` je odvozená od třídy `Robot` a uvědomí si, co to v tomto konkrétním případě znamená.

Poznámky

- Na této závěrečné úloze mohou studenti demonstrovat všechny předchozí cíle.
- Je tu také možnost studety názorně seznámit s další vlastností OOP – dědičností. Třída `RobotTuzka` je odvozená od třídy `Robot`, dědí tedy všechny její vlastnosti a metody. Rozšiřuje je o metody umožňující kreslit. Instance třídy `RobotTuzka` mohou tedy vždy zastoupit instance třídy `Robot`, ale obráceně ne. Vzájemný vztah mezi třídami, tedy i dědičnost, zobrazujeme v modelovacím jazyku UML pomocí diagramu tříd (obr. 2.25) [4].



Obrázek 2.25: Třída `RobotTuzka` je odvozená od třídy `Robot`

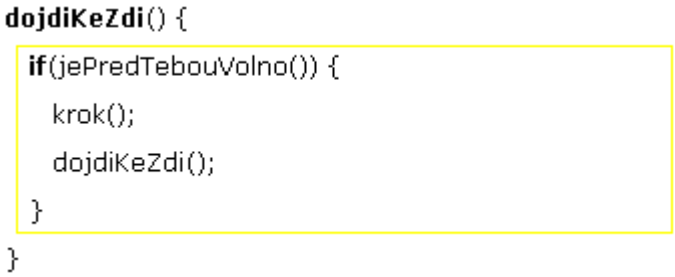
2.3.16 Možnosti rekurze

Při řešení některých úloh lze poměrně úspěšně a hlavně názorně užít rekurzi. Názorně proto, že se metody rozbalují přímo v kódu, řízení programu nikam neskáče a metody jsou barevně ohraničené. Rekurzivně volaná metoda je tedy názorně vnořená. Ukážeme si to na dvou příkladech přímé rekurze.

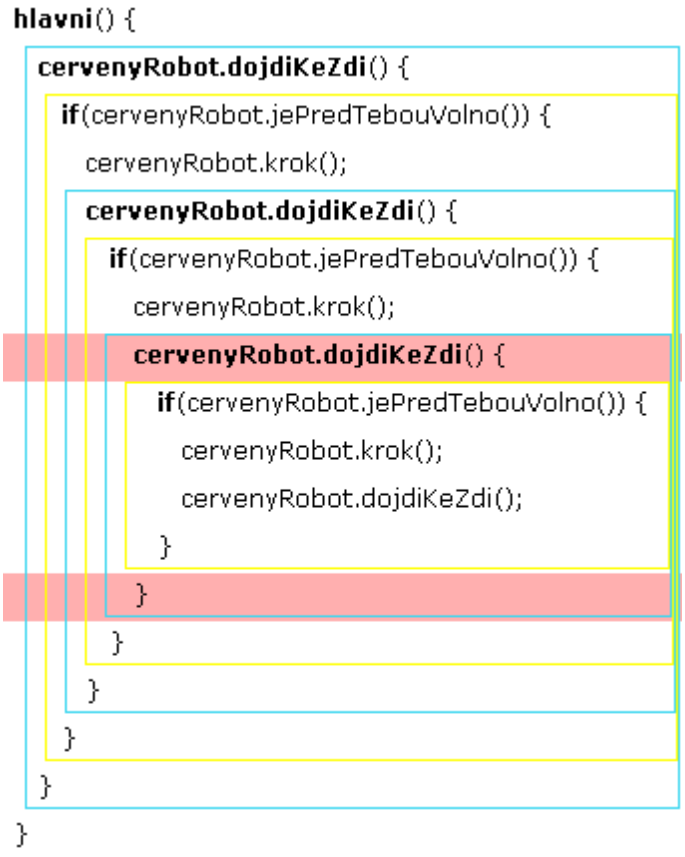
Již známou metodu robota `dojdiKeZdi()` můžeme navrhnout i tak, jak je vidět na obrázku 2.26, na obrázku 2.27 je vidět při krokování. Bohužel v tomto prostředí a u této metody není názorně vidět vynořování, dílčí metody prostě skončí, a tím skončí i program.

Takže se podíváme na druhou ukázku, na metodu `najdiPulku()`, kterou lze úspěšně použít v několika předchozích úkolech. Myšlenka je prostá, cestou ke zdi dělej dva kroky, cestou zpátky jeden krok. Metodu vidíte na obrázku 2.28 a na obrázku 2.29 je vidět při vynořování, robot bude právě provádět krok vzad, a pak udělá ještě jeden krok vzad.

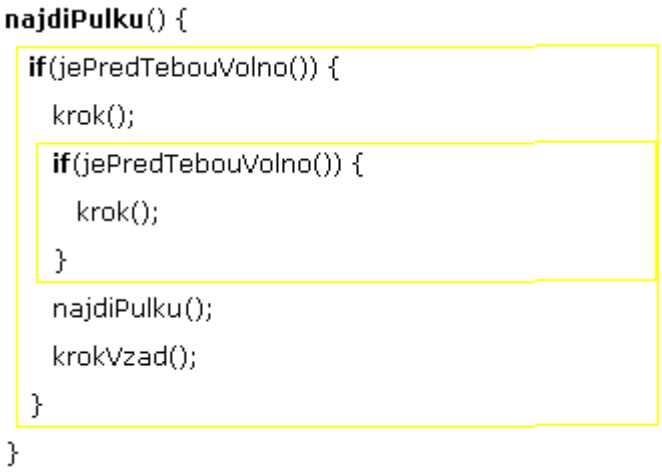
Z pohledu návrhu algoritmu je vhodné upozornit na požadavek jeho konečnosti. Jak víme, v programu je nekonečná rekurze v podstatě nepřípustná.



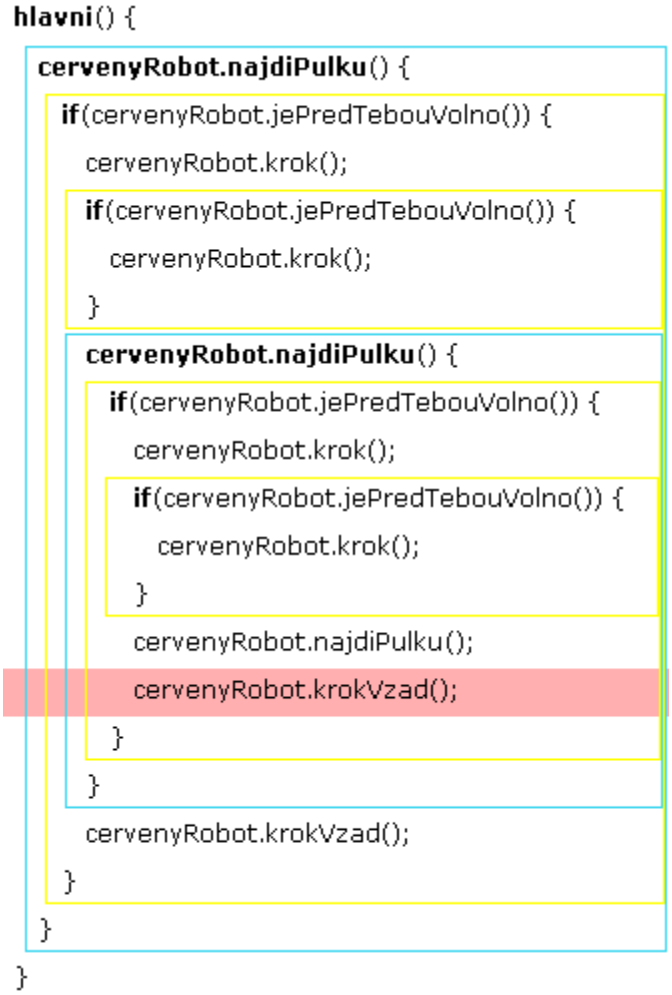
Obrázek 2.26: Metoda `dojdiKeZdi()` s využitím rekurze



Obrázek 2.27: Rekurzivně volaná metoda `dojdiKeZdi()` při krokování



Obrázek 2.28: Metoda `najdiPulku()` s využitím rekurze



Obrázek 2.29: Rekurzivně volaná metoda `najdiPulku()` při krokování

Kapitola 3

Uživatelská příručka programu RUR

3.1 Možnosti spuštění programu

Všechny základní soubory aplikace jsou sbaleny do jednoho archivu – `RUR.jar`. Jednotlivé úlohy jsou v datových souborech s příponou `.rur`.

Výhodou je, že soubor `RUR.jar` lze spouštět na jakékoliv platformě, pro kterou máme k dispozici příslušný virtuální stroj jazyka Java JVM, který je součástí JRE (Java Runtime Enviroment), případně SDK (Standart developement kid).

Předpokládá se tedy, že na počítači je nainstalované J2SE 1.4.1 JRE, či novější, ve verzi toho operačního systému, ve kterém je program spouštěn. Pro různé operační systémy si lze zdarma stáhnout konkrétní instalace JRE z internetových stránek `<http://java.sun.com/>`. Pro operační systém Windows lze využít instalaci přiloženou na CD s programem. Při instalaci postupujeme dle instrukcí.

Program se spustí standardně stejně jako jiné programy v jazyce JAVA. V operačních systémech Windows máme tyto možnosti:

1. Standardně pomocí ikonky souboru RUR.jar. Po spuštění se načte poslední úloha, pokud je k dispozici, případně lze v aplikaci načíst novou úlohu.
2. V příkazovém řádku, spustíme `java -jar RUR.jar`, případně s parametrem `java -jar RUR.jar konkretniUloha.rur`.
3. Po asociaci souborů a úpravě příslušných klíčů v registrech operačního systému se program spustí pomocí ikonky konkrétní úlohy, tedy datového souboru s příponou `.rur`.

Po ukončení běhu aplikace se aktuální úloha a nastavení uloží do souboru `last.rur`. Soubor se vytvoří ve stejném adresáři jako archiv `RUR.jar`. A pokud ho tam aplikace po spuštění nalezne, tak se uložená úloha zobrazí a vlastnosti nastaví.

HW nároky aplikace nejsou vysoké, obecně se dá říci, že vyhovují běžně užívané osobní počítače.

3.2 Rozvržení hlavního okna programu

Veškeré ilustrující obrázky jsou ukázky zobrazení aplikace ve Windows XP se standardním nastavením. V jiných operačních systémech bude popisované prostředí více či méně graficky odlišné, ale funkce jednotlivých částí bude stejná. Grafika se také bude mírně odlišovat při použití různých verzí JRE. Na zmenšených obrázcích si můžete prohlédnout zobrazení jedné z počátečních úloh kurzu. Například na obrázku 3.1 je hlavní okno aplikace zobrazené v operačním systému Microsoft Windows XP se standardním nastavením a obrázek 3.2 ilustruje zobrazení v systému Linux v prostředí KDE 3.5.1 se standardním nastavením.

Hlavní okno programu je vertikálně rozděleno pomocí dvou oddělovačů na tři části, jejich šířku lze volit pomocí myši. Jednotlivé části budeme zleva nazývat Panelem animace, Panelem kódu a Panelem nabídky.

Nahoře každé části okna jsou panely nástrojů s tlačítky. Jednotlivé panely nástrojů lze pomocí myši vyjmout a přemístit na jinou stranu své části okna či je nechat jako plovoucí. U tlačítek se zobrazuje informační titulek, který společně s ikonkami přispívá k intuitivnímu ovládání. Detailní popis funkce jednotlivých tlačítek naleznete v kapitole Funkce jednotlivých tlačítek.

3.2.1 Animační panel

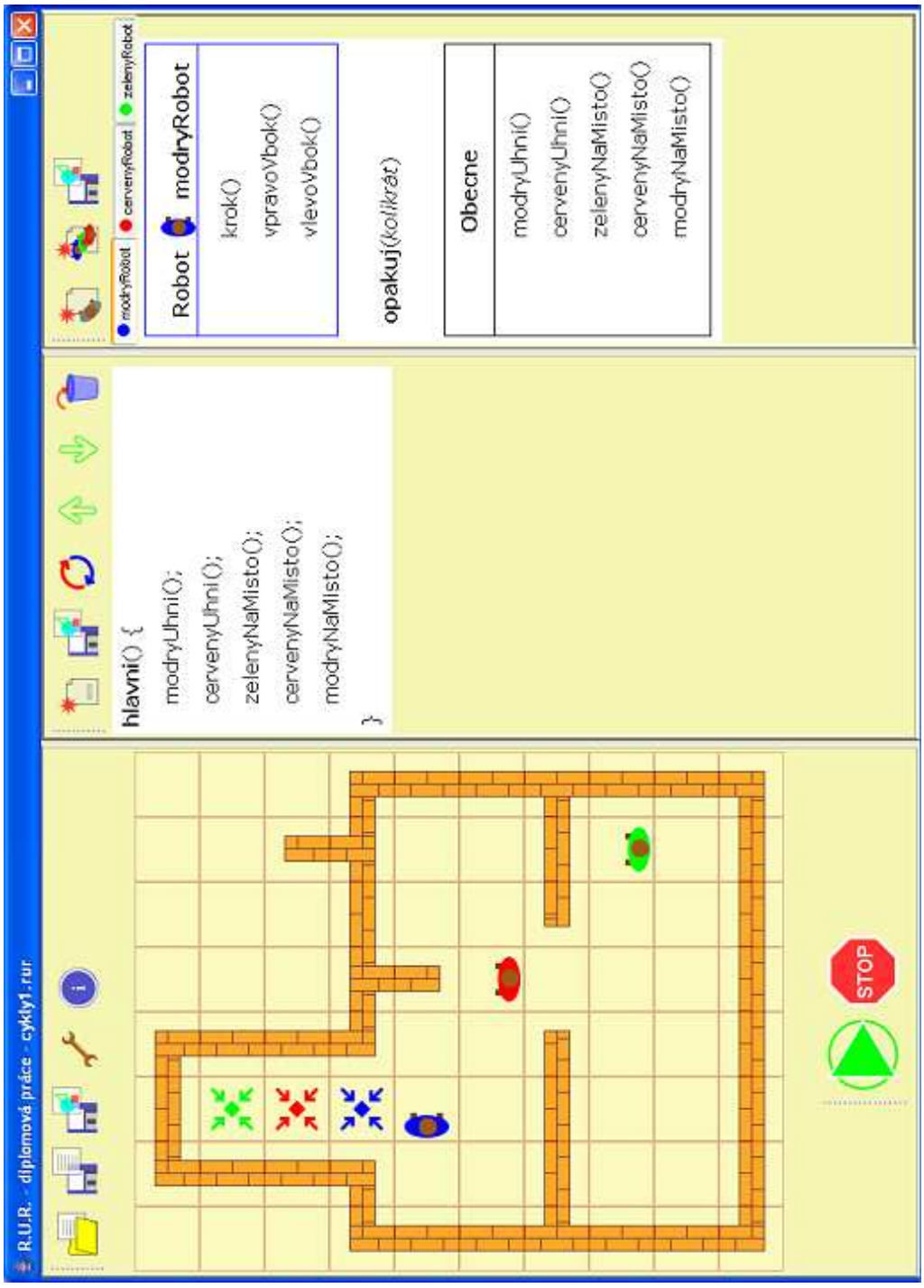
Levý panel hlavního okna nazveme Animační panel, neboť uprostřed se zobrazuje aktuální animační plocha. Velikost obrázku plochy se automaticky přizpůsobuje velikosti panelu.

Nahoře se nachází panel nástrojů s tlačítky (horní část obrázku 3.3), jedná se o tlačítka:

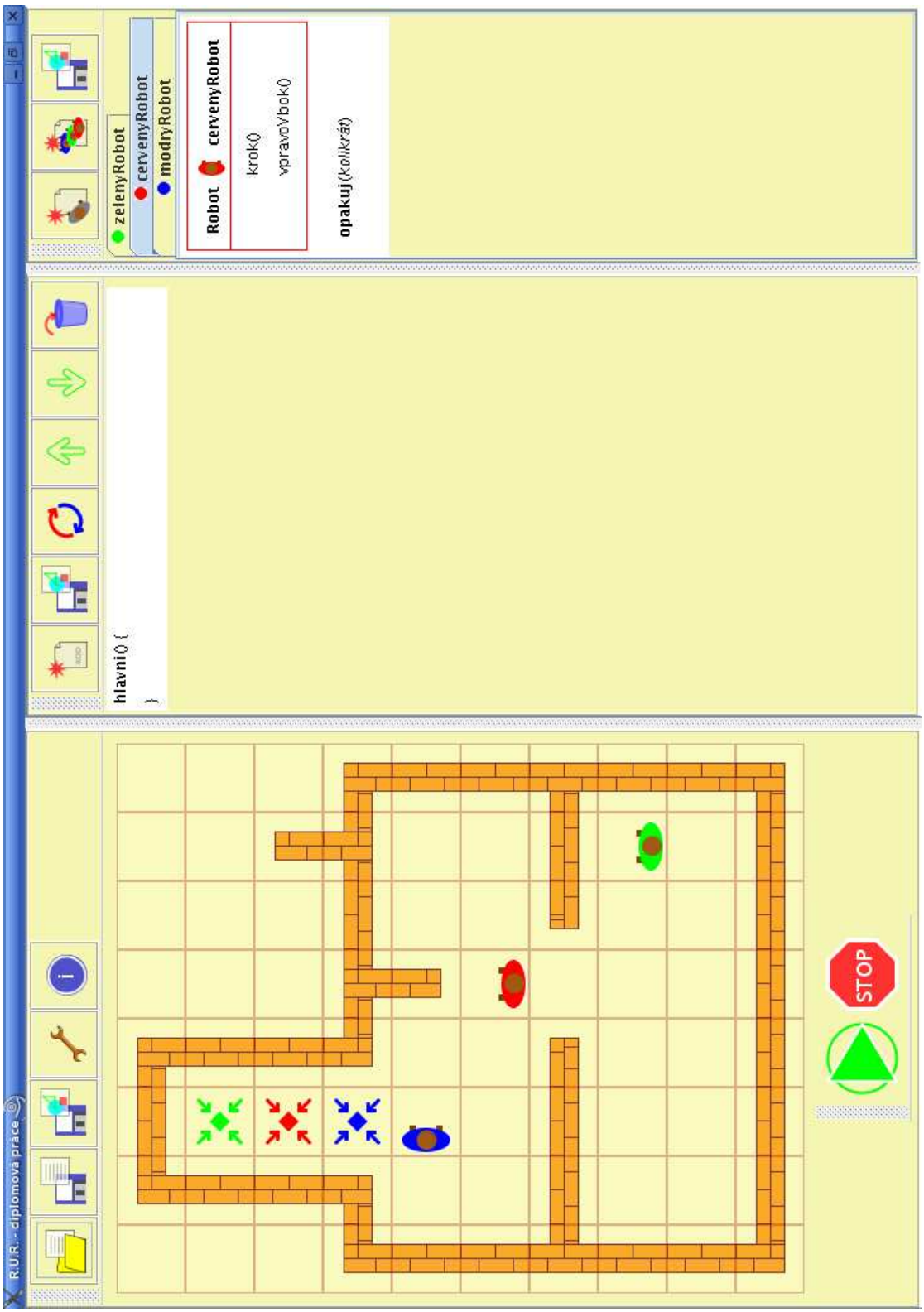
- Načíst úlohu ze souboru.
- Uložit aktuální úlohu.
- Uložit aktuální obrázek animační plochy.
- Nastavit vlastnosti.
- Info o programu.

Mezi horními tlačítky a animační plochou se u některých úloh zobrazuje rozvírací seznam, který umožňuje vybrat dílčí konfiguraci úlohy či její dílčí vlastnosti. Na obrázku 3.3 je příklad z úlohy, kde zvolený algoritmus musí být dostatečně obecný, aby fungoval pro různě velké náhodně volené čtvercové plochy a náhodné umístění robota. Případně si také ještě můžeme vybrat zachování naposledy použitého rozměru plochy a umístění robota.

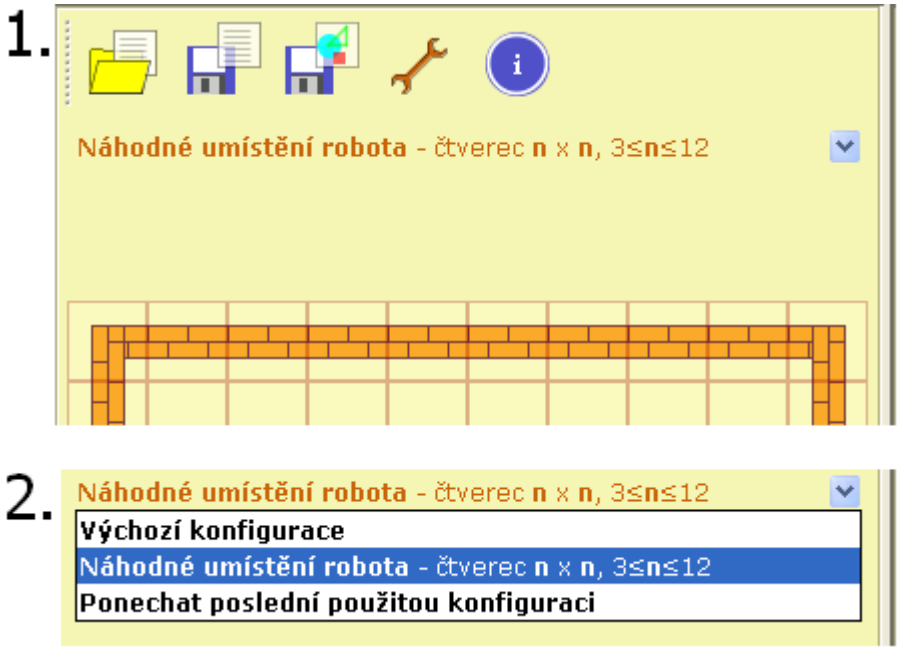
Dole je panel nástrojů s tlačítky `Krok` a `Stop` (viz obr. 3.4), umožňující krokování programu. I tento panel nástrojů může být plovoucí. Pozor, pokud umístíme horní panel nástrojů na stejnou stranu, může dojít k jejich překrytí.



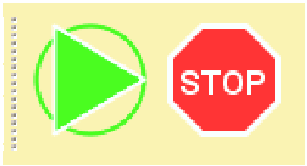
Obrázek 3.1: Hlavní okno aplikace ve Windows XP



Obrázek 3.2: Hlavní okno aplikace v Linuxu



Obrázek 3.3: Příklad zobrazení a výběru možnosti konfigurace dané úlohy

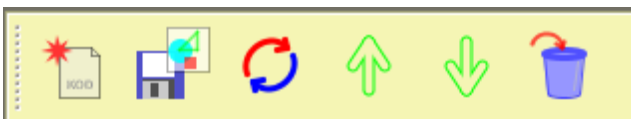


Obrázek 3.4: Tlačítka `Krok` a `Stop`

3.2.2 Panel kódu

Prostřední panel zobrazuje instrukce hlavní metody, případně jiných vámi vytvořených právě upravovaných metod. Každá metoda se může zobrazit na samostatné záložce.

Nahoře se nachází panel nástroj s tlačítky (viz obr. 3.5), který se vztahují k právě zobrazené metodě, případně vybrané instrukci kódu.



Obrázek 3.5: Panel nástrojů s tlačítky Panelu kódu

Jedná se o tlačítka:

- Odstranit všechny instrukce právě zobrazené metody.
- Uložit obrázek kódu právě zobrazené metody.

- Upravit (vybranou metodu či programovou konstrukci).
- Posunout vybranou instrukci o jeden řádek nahoru.
- Posunout vybranou instrukci o jeden řádek dolů.
- Odstranit (z kódu vybranou instrukci).

3.2.3 Panel nabídky

V pravé části okna se zobrazují instrukce – metody jednotlivých robotů, aktuální programové konstrukce a obecné metody. Jedná se o nabídku, z níž sestavujeme program. Každý robot, vyskytující se v dané úloze, má svou záložku. Při úpravě vámi vytvořené metody robota se zobrazuje diagram příslušné třídy.

I zde se nahoře nachází panel nástrojů (viz obr. 3.6) s tlačítky:

- Vytvořit novou metodu robota.
- Vytvořit novou obecnou metodu.
- Uložit aktuální obrázek právě zobrazené nabídky.

Obrázek 3.6: Panel nástrojů s tlačítky Panelu nabídky

3.3 Ovládání programu

Ovládání je upraveno pro přímé použití na interaktivní dotykové tabuli. Je tu tedy snaha o intuitivní ovládání pomocí levého tlačítka myši, což odpovídá stisknutí tabule, většinou prstem. Klávesnice je potřeba pouze při přejmenování vámi vytvořené metody nebo pojmenování ukládané úlohy, kdy použijeme klávesnici počítače nebo si zobrazíme klávesnici dotykové tabule. Pravé tlačítko myši se nemusí používat vůbec.

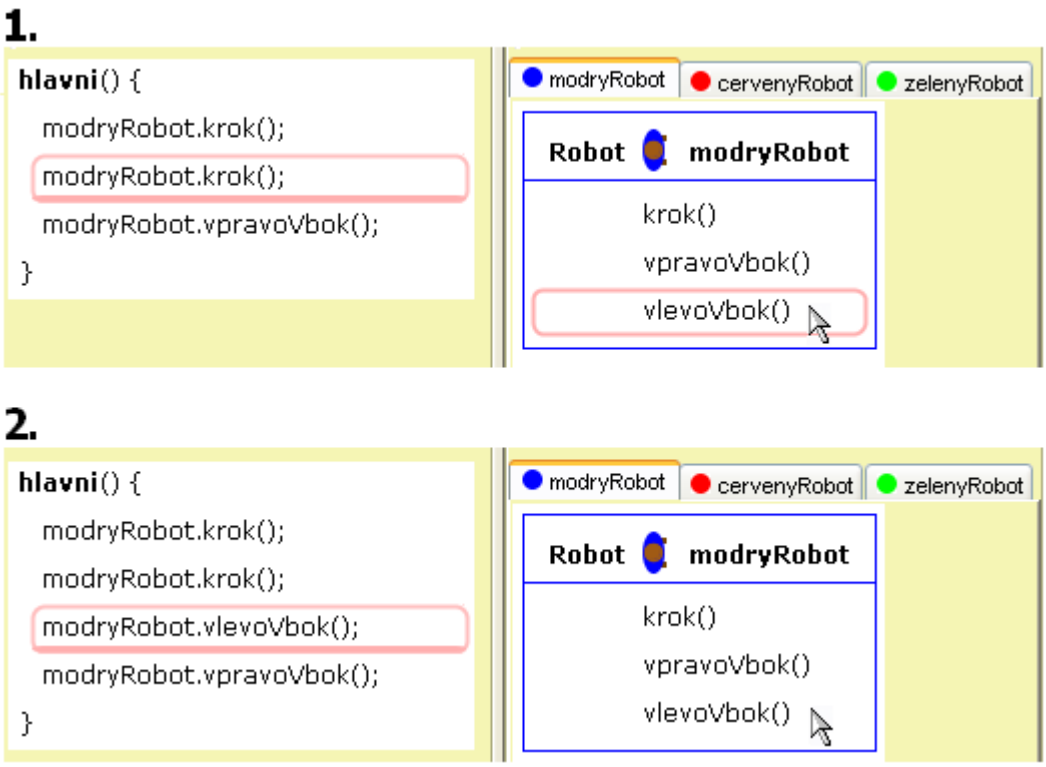
Program se může nacházet ve dvou základních módech. Jednak můžeme programovat, tedy upravovat kód. A jednak lze vytvořený kód krokovat a sledovat provádění jeho instrukcí na animační ploše.

Jak už bylo naznačeno, v této aplikaci se programuje velmi jednoduše. Jednotlivé instrukce se nezapisují na klávesnici, ale pouze stisknutím tlačítka myši vybíráme z nabídky kam a která instrukce má být vložena. Nejprve se v následujících kapitolách seznámíme s tím, jak instrukci vložit do kódu, případně ji upravit, přesunout či odstranit, jak pracovat s vámi vytvořenými metodami. Pak si popíšeme, jak program krokovat a nakonec jak úlohu načíst a uložit.

3.3.1 Vložení instrukce do kódu právě zobrazené metody

Pokud program není zrovna krokován, lze jakoukoliv instrukci z aktuální nabídky vložit jednoduchým stisknutím a uvolněním levého tlačítka myši. Instrukce se vždy vloží za právě vybranou instrukci v kódu zobrazené metody. Vybraná instrukce v kódu je zvýrazněna růžovým oválem a vybereme ji prostým kliknutím myši.

Například na obrázku 3.7 můžeme sledovat ukázku vkládání instrukce nabídky `modryRobot.vlevoVbok()` za vybranou instrukci `modryRobot.krok()` v hlavní metodě.



Obrázek 3.7: Ukázka vkládání instrukce do kódu

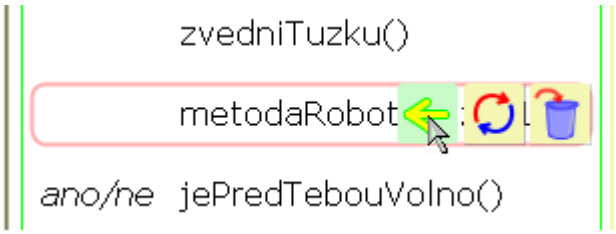
Nejprve (1.) je zobrazena situace ve chvíli, kdy máme v kódu hlavní metody (vlevo) vybranou instrukci `modryRobot.krok()`, je tedy ohraničena růžovým oválem. Následně stiskneme tlačítko myši nad vkládanou instrukcí nabídky `vlevoVbok()`, tato instrukce se po dobu stisknutí myši také růžově ohraničí (vpravo).

Poté (2.) vidíme situaci po uvolnění tlačítka myši, tedy po vložení zvolené instrukce. Vložená instrukce je na požadovaném místě a je ohraničena růžovým oválem, další instrukce se tedy případně přidá za tuto právě vloženou instrukci.

Vkládání vámi vytvořených metod

Vámi vytvořenou metodu, obecnou i metodu robota, vložíme do kódu stejně jako jinou instrukci, tedy kliknutím myši. Při stisknutí tlačítka myši nad zvolenou metodou nabídky se však zobrazí i tlačítka. První zobrazené tlačítko se objeví právě

pod kurzorem a bude stisknuté (obr. 3.8). Pokud tlačítko myši uvolníme a první zobrazené tlačítko bude stlačené, tak se metoda přidá standardně do kódu, nemusíme s myší tedy vůbec pohybovat. Pokud však při stále stlačeném tlačítku myši vybereme jiné zobrazené tlačítko, provede se odpovídající akce. Jestliže tlačítko myši uvolníme mimo zobrazená tlačítka, nestane se nic.



Obrázek 3.8: Nabídka tlačítek při vkládání vytvořené metody do kódu

Tento netradiční postup byl zvolen pro snadnější ovládání, metoda se přidá stejně jednoduše jako jiné instrukce pouhým kliknutím. Jinou akci (upravit, odstranit) vybereme stejně snadno. Tudíž je jednoduší ovládání na dotykové tabuli, kde nemusíme navolit použití pravého tlačítka, které bývá zvykem používat pro zobrazení nabídky a dalšího možného výběru. Stačí při výběru udržet kontakt s plochou tabule, nepoklepávat.

Vkládání programových konstrukcí

Programové konstrukce (cykly `opakuj`, `while` a podmíněná instrukce `if-else`) se do kódu právě zobrazené metody vloží stejně jako jiné instrukce, kliknutím myši. Zároveň se však zobrazí i dialog umožňující nastavení vlastností. Jednotlivá dialogová okna a možnosti jsou popsány v části Změna vlastností programových konstrukcí. Dílčí instrukce do bloku těchto konstrukcí vložíme standardně, instrukce se vždy vloží na řádek pod růžový ovál ohraničují vybranou instrukci právě zobrazené metody. Blok instrukcí dané programové konstrukce je ohraničen složenými závorkami a pro větší přehlednost a názornost barevně ohraničen, což je vidět například na obrázcích na straně 67.

3.3.2 Přesunutí vybrané instrukce v kódu právě zobrazené metody

Aplikace umožňuje přesouvat vybrané instrukce pouze uvnitř právě zobrazené metody. Nelze přesouvat instrukce z jedné metody do jiné metody, například rozhodnete-li se z části kódu vytvořit novou metodu.

Přesouvá se instrukce, která je aktuálně vybraná, tedy ohraničená růžovým oválem, a lze ji posunout vždy o jeden řádek kódu nahoru nebo dolů, tedy vyměnit dvě nad sebou zobrazené instrukce. Posunutí nahoru či dolů se provede po klinutí na odpovídající tlačítko na panelu nástrojů Panelu kódu (viz obr. 3.5 na straně 50). Přičemž se může jednat o celé bloky kódu v případě, že vybereme blok instrukcí programové konstrukce, např. blok instrukcí cyklu `while`.

V podstatě můžeme instrukci přesunout, v rámci jedné metody, kamkoliv. Postup je vhodný, pokud potřebujeme přesunout jeden blok instrukcí do bloku jiné programové konstrukce a nechceme danou část volit znovu. Lze samozřejmě instrukci z kódu odstranit a znovu vložit na požadované místo, můžete tím předejít zbytečným chybám.

3.3.3 Odstranění vybrané instrukce z kódu právě zobrazené metody

Instrukci, kterou chceme z kódu odstranit, nejprve vybereme prostým kliknutím myši, bude tedy ohraničená růžovým oválem. Vybranou instrukci nevratně odstraníme kliknutím na tlačítko `Odstranit`, jedná se o poslední tlačítko panelu nástrojů Panelu kódu (viz obr. 3.5 na straně 50).

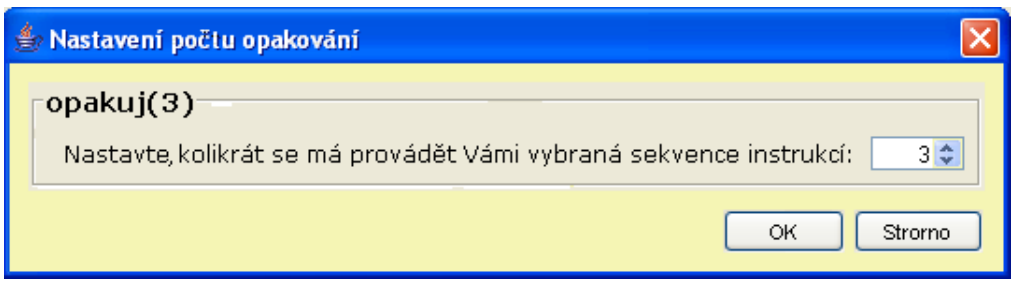
Případně je možné odstranit všechny instrukce právě zobrazené metody najednou, jestliže stisknete první tlačítko na panelu nástrojů Panelu Kódu (viz obr. 3.5 na straně 50), a tudíž začneme s čistým listem. Protože se jedná o nevratnou změnu, program před odstraněním nabízí možnost uložení celé úlohy. Možnost uložení nenabídne, pokud od posledního uložení nebyly provedeny žádné změny nebo je metoda prázdná.

3.3.4 Změna vlastností vložených programových konstrukcí

Požadovanou programovou konstrukci v kódu vybereme kliknutím myši, ohraničí se růžovým oválem. Poté klikneme na třetí tlačítko panelu nástrojů Panelu kódu `Upravit`(viz obrázek 3.16 na straně 59). Postup je stejný jako při požadavku na úpravu metody, což ilustruje obrázek 3.15 na straně 59. Po klinutí na tlačítko se nám zobrazí příslušné dialogové okno. Stejného efektu docílíme přidáním zvolené programové konstrukce do kódu.

Cyklus `opakuj(`*`kolikrat`*`)`

U cyklu `opakuj` je nutné nastavit počet opakování – kolikrát se bude sekvence instrukcí v bloku cyklu opakovat (viz obrázek 3.9). Přednastavena je nula – cyklus se neprovede ani jednou.



Obrázek 3.9: Dialogové okno nastavení počtu opakování cyklu `opakuj`

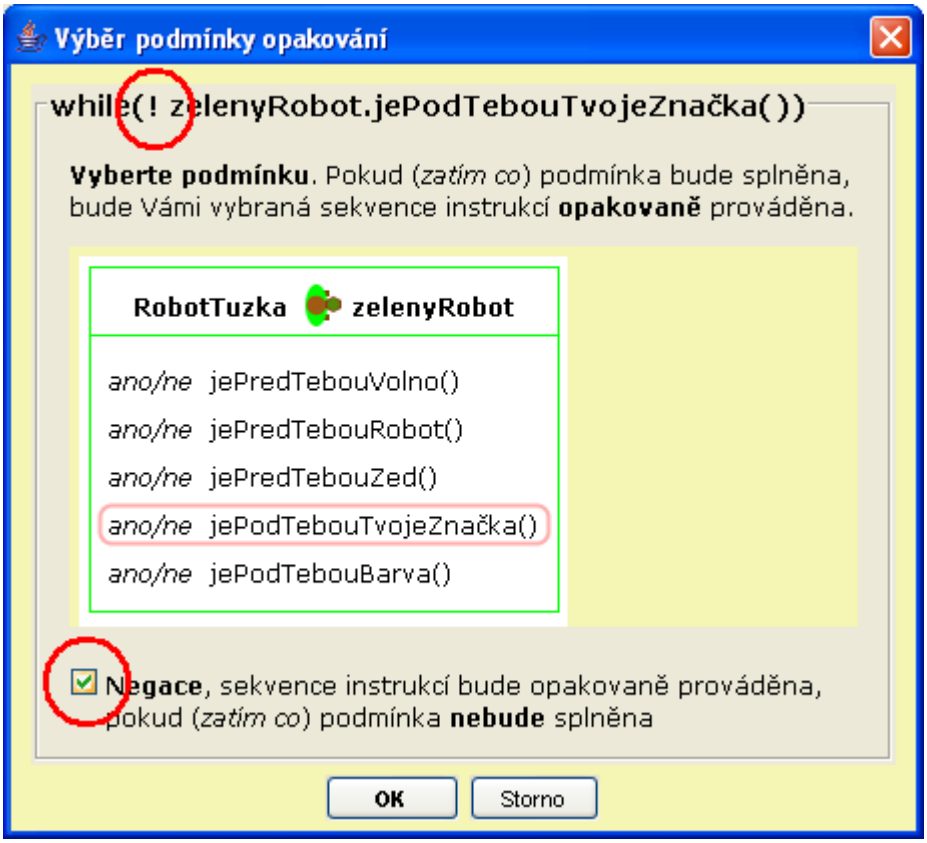
Cyklus `while(`*podmínka*`)`

U cyklu `while` je nutné nastavit podmínku opakování. Při splnění podmínky (zatím co bude splněna) bude sekvence instrukcí v bloku cyklu opakovaně prováděna. Podmínku nastavíme kliknutím myší, vybraná podmínka se zobrazí i v náhledu, v titulku ohraničení.

Nebude-li žádná podmínka vybrána, je přednastavena hodnota `true` – pravda, cyklus tedy bude prováděn neustále, jako by podmínka byla splněna. Pokud můžeme volit mezi několika roboty, jejich podmínky jsou zobrazeny na záložkách, stejně jako na Panelu nabídky, obdobně jako tomu je při volbě podmínky bloku podmíněných instrukcí na obrázku 3.11.

V dialogovém okně máme možnost si zvolit i negaci podmínky. V tom případě bude výsledek podmínky negován a instrukce budou opakovaně prováděny pokud (zatím co) podmínka nebude splněna.

Například na obrázku 3.10 je vybrána podmínka `jePodTebouTvojeZnacka()` a zároveň je vybrána negace podmínky, tudíž se před podmínkou zobrazuje vykřičník.

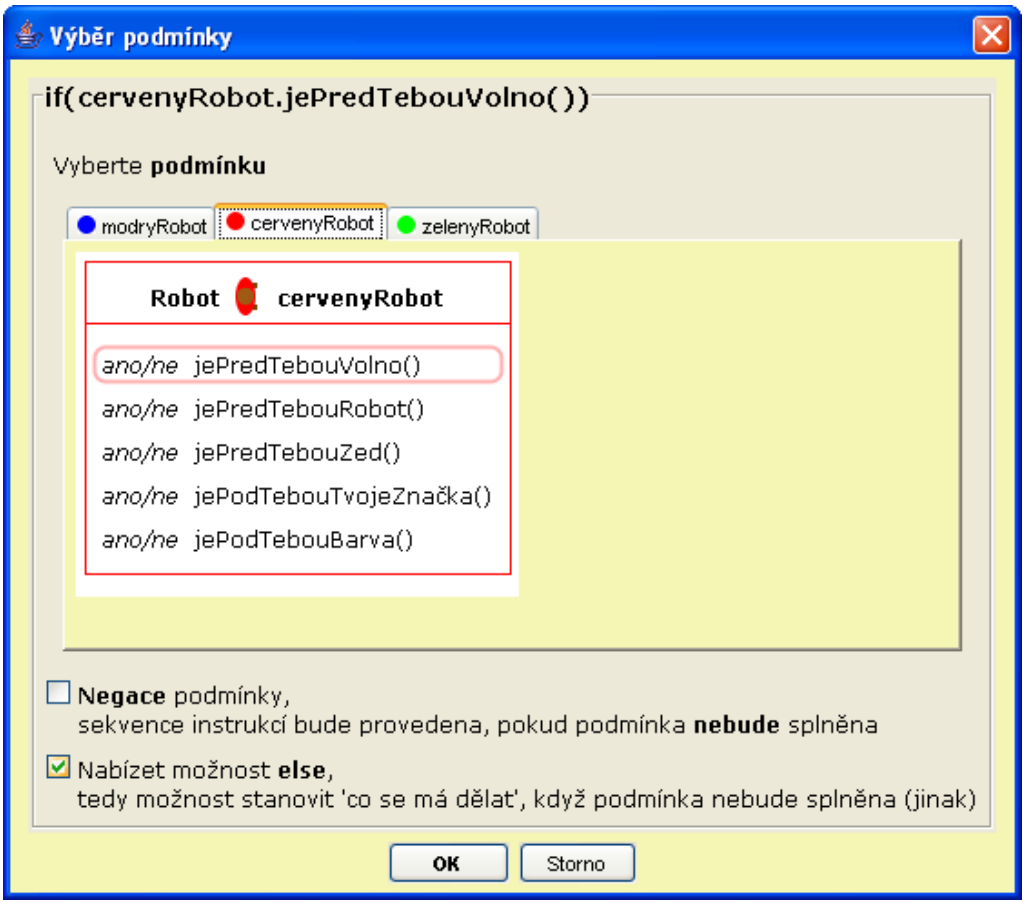


Obrázek 3.10: Dialogové okno pro nastavení vlastností cyklu `while`

Blok podmíněných instrukcí `if-else`

I zde je nutné nastavit podmínku, blok instrukcí se provede právě tehdy, jestliže (if) bude podmínka splněna. Ale máme možnost také určit, co se má dít, když podmínka splněna nebude – jinak (else) proveď tento blok instrukcí.

Postup výběru podmínky je obdobný, včetně možnosti negace, jako u cyklu `while` Ukázku dialogového okna umožňujícího výběr podmínky je vidět na obrázku 3.11, kde se dá volit mezi několika roboty, jejichž možné podmínky jsou zobrazeny na odpovídajících záložkách. Právě je vybrána podmínka červeného robota `cervenyRobot.jePredTebouVolno()`, což se zobrazuje i v titulku ohraničení.



Obrázek 3.11: Dialogové okno volby vlastností podmíněné instrukce

V dolní části dialogového okna lze volit, zda-li se má v kódu nabízet možnost `else`. Zda-li bude tato varianta primárně zaškrtnuta, určuje konfigurace dané úlohy. Volba není aktivní, jestliže jsou v kódu do bloku pod `else` již vloženy instrukce. Na obrázku 3.12 vlevo je zobrazena ukázka části kódu s možností `else`, vpravo jednoduší alternativa – bez `else`.

```
if(cervenyRobot.jePredTebouVolno()) {
  cervenyRobot.krok();
}else{
  cervenyRobot.vpravoVbok();
}
```

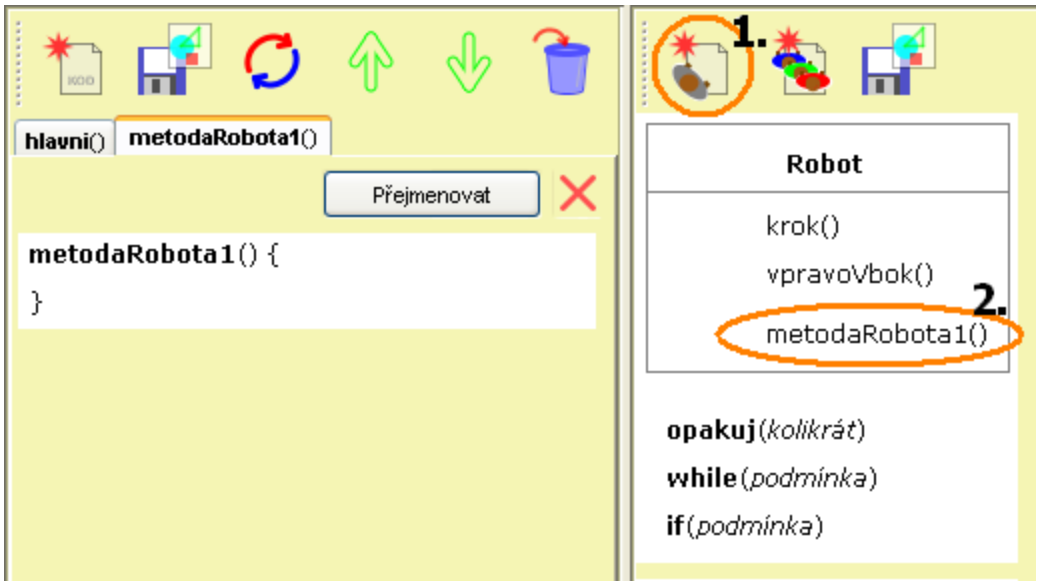
```
if(cervenyRobot.jePredTebouVolno()) {
  cervenyRobot.krok();
}
```

Obrázek 3.12: Ukázka možností podmíněné instrukce `if-else`

3.3.5 Vytvoření nové metody

Novou metodu vytvoříme snadno, klikneme na příslušné tlačítko na panelu nástrojů Panelu nabídky (viz obr. 3.6 na straně 51). Buď vytvoříme novou metodu robota, nebo novou obecnou metodu.

Nová metoda robota se obrazně přidá do nabídky jako další metoda třídy `Robot`, případně `RobotTuzka` (obr. 3.13). Laicky řečeno, naučíme robota vykonávat další činnost skládající se z posloupnosti již známých instrukcí. Naučíme to jednoho robota a budou to hned umět všichni roboti ze stejné třídy. Záleží na tom, která záložka na Panelu nabídky byla při požadavku na vytvoření metody aktivní. Pokud to byla záložka obyčejného robota (instance třídy `Robot`), pak vytvořenou metodu budou moci využívat všichni roboti. Ale pokud to byl robot s tužkou (instance třídy `RobotTuzka`), budou ji mít k dispozici jen roboti s tužkou. Neboť Robot s tužkou je speciálním případem obyčejného robota, umí navíc kreslit a jeho metody by obyčejný robot nemusel umět vykonávat. Odborně řečeno, třída `RobotTuzka` je odvozená od třídy `Robot`, dědí tedy všechny metody předka, přičemž v této aplikaci nelze zděděné metody přepisovat.



Obrázek 3.13: Vytvoření nové metody robota

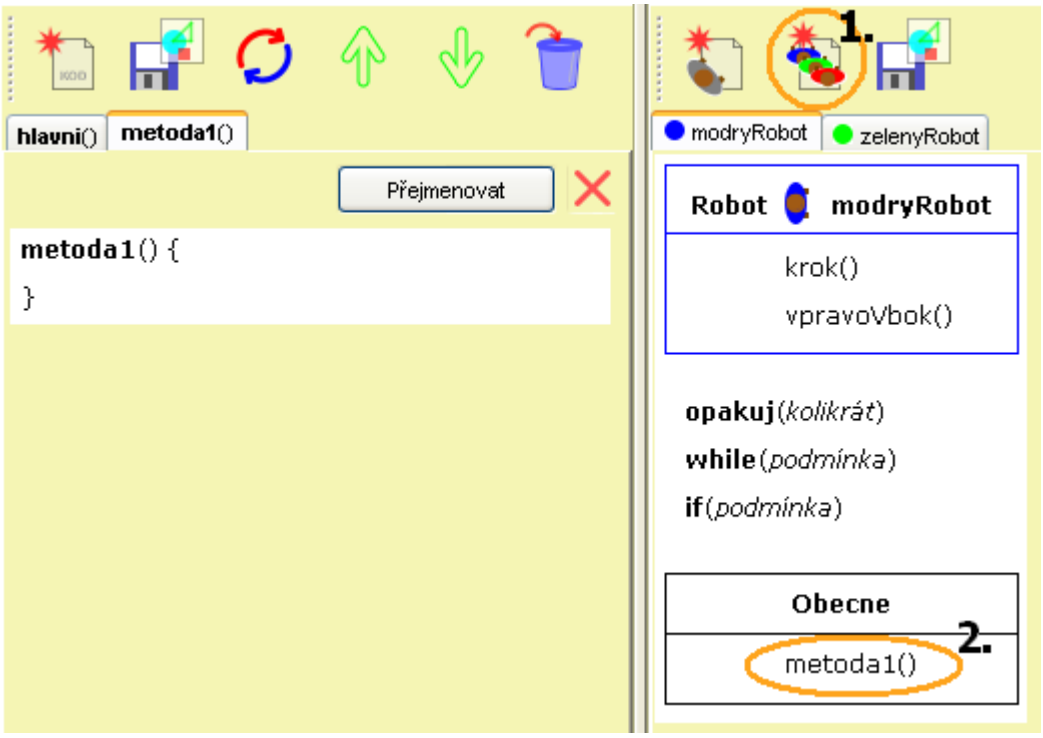
Nová obecná metoda se obrazně přidá do třídy `Obecne` (obr. 3.14). Jedná se tedy o obecnou metodu, ve které definujeme, co který konkrétní robot (konkrétní instance) má dělat.

Po vytvoření se nová metoda robota zobrazí mezi jeho instrukcemi, nová obecná metoda se zobrazí pod instrukcemi robota. V nabídce jsou naznačeny diagramy příslušných tříd.

Nová metoda se pojmenuje tak, aby nedocházelo k duplicitě názvů. A zobrazí se na své záložce na Panelu kódu, kde ji můžeme upravovat a vhodně přejmenovat podle činnosti, kterou bude metoda (robot) vykonávat. Postup přejmenování metody naleznete v následující části této příručky. Přidávání instrukcí do metody popisuje část Vložení instrukce do kódu právě zobrazené metody.

Na obrázku 3.13 je příklad situace po stisknutí tlačítka `Vytvořit novou metodu robota` (1.), metoda se pojmenuje `metodaRobota1`, přidá se do třídy `RobotTuzka` (2.) a zobrazí se její záložka na Panelu kódu.

Obrázek 3.14 ilustruje situaci po stisknutí tlačítka `Vytvořit novou obecnou metodu robota` (1.), metoda se pojmenuje `metoda1`, obrazně se přidá do třídy `Obecne` (2.) a zobrazí se její záložka na Panelu kódu.



Obrázek 3.14: Vytvoření nové obecné metody

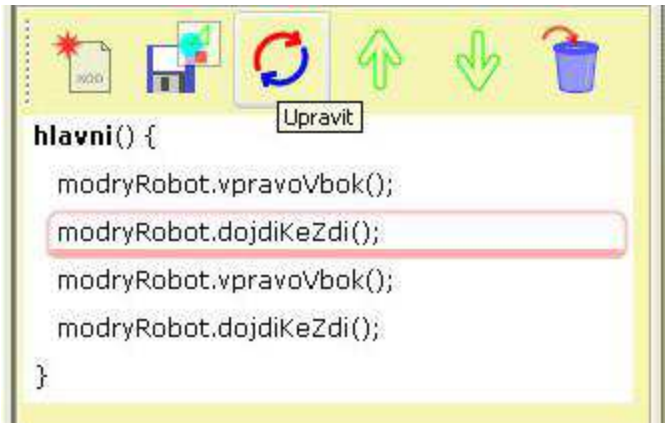
3.3.6 Úprava vytvořené metody

Metoda musí být nejprve zobrazena na své záložce Panelu kódu, kde je možno ji upravovat. Pokud ještě není zobrazena, jsou tři možnosti jak toho docílit:

1. Záložka metody se automaticky zobrazí při vytvoření metody (viz obrázek 3.17 na straně 60).
2. Již vytvořenou metodu můžeme upravovat. Když na Panelu nabídky nad zvolenou metodou stiskneme levé tlačítko myši, držíme ho stále stisknuté, tak se na panelu zobrazí nabídka tlačítek. Stále držíme stisknuté tlačítko a vybereme prostřední zobrazené tlačítko `Upravit` (viz obrázek 3.15).

Obrázek 3.15: Upravit vybranou metodu z nabídky

3. Pokud jsme již metodu přidali do kódu jiné metody na Panelu kódu, vybereme v zobrazeném kódu upravovanou metodu stisknutím myši, zvýrazní se růžovým oválem. Poté stiskneme na panelu nástrojů Panelu Nabídky tlačítko `Upravit`(viz obrázek 3.16).



Obrázek 3.16: Upravit vybranou metodu

Pro metodu se vytvoří na Panelu kódu stejně pojmenovaná záložka (viz obrázek 3.17) a metodu poté můžeme:

- přejmenovat,
- vkládat do ní další instrukce,
- odstraňovat instrukce,
- měnit pořadí jednotlivých instrukcí,

- upravovat vlastnosti v metodě použitých programových konstrukcí,
- otestovat (odkrokovat) ji, jedná-li se o obecnou metodu, samozřejmě také záleží na dané konfiguraci animační plochy.

Jakékoliv úpravy jsou ihned nevratně provedeny a projeví se ve všech výskytech dané metody.

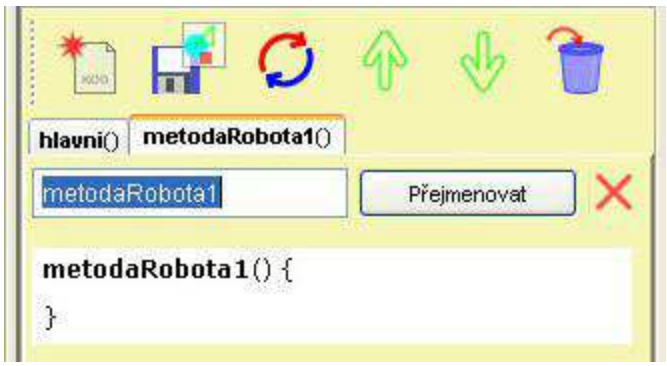
Obrázek 3.17: Záložka upravované metody

Přejmenování vytvořené metody

Přejmenovat lze kteroukoliv vámi vytvořenou metodu. Nejprve však musí být zvolená metoda zobrazena na záložce Panelu kódu, kde je ji možno upravovat. Postup jak zvolenou metodu zobrazit na Panelu kódu naleznete na počátku kapitoly Úprava vytvořené metody.

Je potřeba si uvědomit, že přejmenováním nevytvoříte novou metodu, přejmenují se všechny její výskyty.

Pokud je záložka metody zobrazena stiskneme tlačítko `Přejmenovat` (viz obrázek 3.17), zobrazí se textové pole, do kterého můžete napsat zvolený název (viz obrázek 3.18).



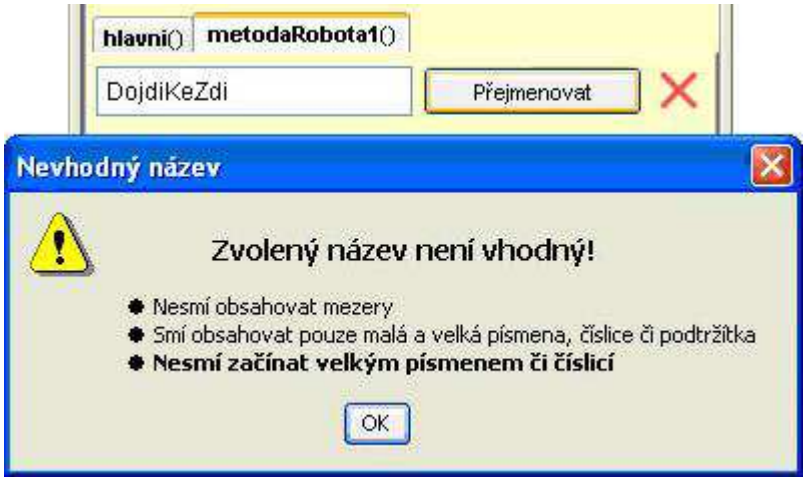
Obrázek 3.18: Přejmenování upravované metody

Pro provedení přejmenování metody, opět stiskneme tlačítko `Přejmenovat`.

Pokud je zvolený název vhodný, metoda se přejmenuje a textové pole zmizí. Před přejmenováním metody se kontroluje vhodnost názvu dle syntaktických pravidel jazyka Java a obecně přijímaných konvencí.

- Název nesmí být již použit,
- neobsahuje mezery,
- nesmí začínat velkým písmenem a číslicí,
- smí obsahovat pouze malá a velká písmena bez diakritiky, číslice a podtržítka.

Pokud je zvolený název vyhodnocen jako nevhodný, zobrazí se zpráva, ve které je tučně zvýrazněno nalezené pochybení (viz obrázek 3.19). Případně se zobrazí zpráva informující o duplicitě názvu.



Obrázek 3.19: Upozornění na nevhodně zvolený název metody

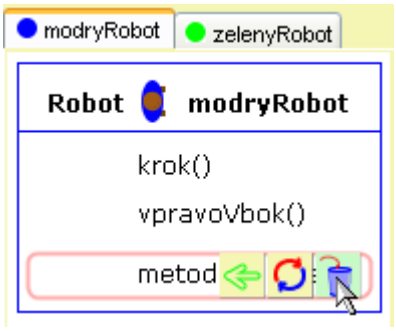
Uzavření záložky upravované metody

Mimo hlavní metody lze kteroukoliv právě aktivní záložku metody uzavřít pomocí tlačítka s ikonkou červeného křížku v pravém horním rohu záložky (viz obrázek 3.17). Metoda není odstraněna, jen zmizí její záložka, všechny provedené úpravy jsou platné. Metodu lze případně dále upravovat po novém zobrazení záložky.

3.3.7 Odstranění vytvořené metody z nabídky

Když na Panelu nabídky stiskneme levé tlačítko myši nad řádkem odstraňované metody a držíme ho stále stisknuté, tak se na panelu v řádku odstraňované metody zobrazí nabídka tlačítek. Stále držíme stisknuté tlačítko a vybereme poslední zobrazené tlačítko `Odstranit` (viz obrázek 3.20).

Metodu není možné z nabídky odstranit, jestliže je odstraňovaná metoda použita v kódu jiné metody (viz obrázek 3.21).

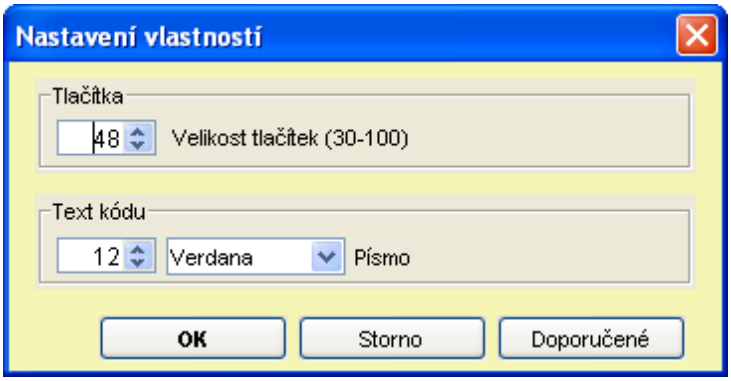


Obrázek 3.20: Odstranění vytvořené metody z nabídky

Obrázek 3.21: Upozornění, metodu nelze z nabídky odstranit

3.3.8 Změna vlastností aplikace

Po stisknutí tlačítka `Nastavení vlastností` na panelu nástrojů Panelu animace (obr. 3.3 na straně 50) se zobrazí dialogové okno (obr. 3.22). Máme možnost nastavit velikost tlačítek a text kódu.



Obrázek 3.22: Dialogové okno nastavení vlastností

Zvolená hodnota velikosti tlačítek stanovuje přímo v pixlech velikost hrany čtvercových tlačítek na horních panelech nástrojů. Tlačítka `Krok` a `Stop` (obr. 3.4 na straně 50) budou 1,5 krát větší. Velikost tlačítek zobrazujících se při kliknutí na vytvořenou metodu na Panelu nabídky (obr. 3.8 na straně 53) je závislá na výšce řádku kódu. Ten je dvakrát větší než je nastavená velikost písma.

Hodnota velikosti písma nastavuje text kódu na Panelu kódu a Panelu nabídky.

Dále máme možnost si zvolit rodinu písma. V nabídce jsou veškerá písma dostupná v používaném operačním systému. Ne všechna jsou vhodná a některá text ani korektně nezobrazí. Ale na druhou stranu, si bez omezení můžeme vybrat dle vlastního vkusu.

Tlačítko `Doporučené` nastaví doporučené hodnoty, tedy velikost tlačítek 48, velikost textu 12 a písmo Verdana. To můžeme akceptovat a nastavit v aplikaci tlačítkem `OK`, nebo stornovat, v tom případě se nastavení těchto hodnot v aplikaci nezmění.

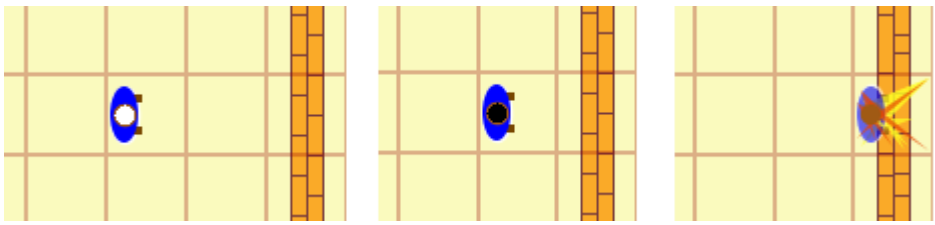
3.3.9 Krokování programu

Aplikace přejde do módu krokování při prvním stisknutí tlačítka `Krok`. Mód krokování se ukončí po stlačení tlačítka `Stop` (viz obrázek 3.4 na straně 50).

Krokovat lze hlavní metodu nebo kteroukoliv obecnou metodu právě zobrazenou na své záložce. Nelze přímo krokovat vytvořenou metodu robota, pokud je zobrazena na aktivní záložce panelu kódu, tehdy není tlačítko `Krok` aktivní. Chceme-li metodu robota otestovat, musíme ji zavolat pro konkrétního robota (instanci dané třídy) v hlavní nebo obecné metodě, kterou krokovat lze.

Po přechodu do módu krokování již nelze kód metod upravovat, příslušná tlačítka a Panel nabídky nebude až do stisknutí tlačítka `Stop` aktivní. Při dalších stisknutích se postupně krokuje ta metoda na záložce, která byla aktivní při prvním stlačení tlačítka (hlavní metoda nebo obecná metoda).

Při následných stlačení tlačítka `Krok` se tedy vykonávají aktuální instrukce programu, vždy ta, jejíž řádek je právě červeně podbarven a příslušné akce se zobrazují na animační ploše. Po ukončení krokování metody setrvá aplikace v módu krokování a tlačítko `Krok` se až do stlačení tlačítka `Stop` zneaktivní. Krokování je ukončeno, když se krokováním dojde na konec algoritmu metody, nebo když robot po nárazu vybuchne (obr. 3.23 vpravo).



Obrázek 3.23: Reakce robota – ano, ne, výbuch

Po stlačení tlačítka `Stop` přejde aplikace z módu krokování programu do módu programování. Provede se restart animační plochy a vlastností robotů, aktivizují se příslušná tlačítka a panely.

Krokování si předvedeme na příkladu. Vytvoříme si metodu robota s tužkou `oteckujPlochu()`, tedy metodu třídy `RobotTuzka`, a otestujeme ji voláním v hlavní metodě pomocí zeleného robota (instance třídy `RobotTuzka`). Při krokování se vždy obrazně zastavíme a popíšeme danou situaci i pomocí odpovídajícího obrázku kódu.

1. Na obr. 3.24 je zobrazena vytvořená metoda robota `oteckujPlochu()`, kterou nelze přímo krokovat, protože nepřísluší žádnému konkrétnímu robotu (instanci), který by ji mohl provádět.

```
obteckujPlochu() {
  dojdiKeZdi();
 while(! jePodTebouBarva()) {
   if(jePredTebouVolno()) {
     nakresliTecku();
     krok();
   }else{
     opakuj(3) {
       vpravoVbok();
     }
   }
 }
}
```

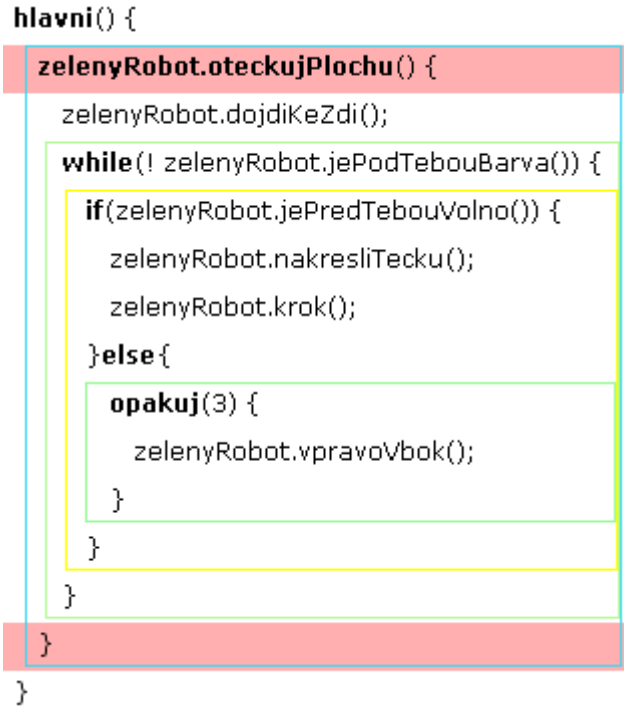
Obrázek 3.24: K příkladu krokování

2. Na obrázku 3.25 v metodě `hlavni()` použijeme zeleného robota (instance třídy `RobotTuzka`) a metodu zavoláme. Teď metodu již krokovat lze, bude ji provádět zelený robot.

```
hlavni() {
  zelenyRobot.oteckujPlochu();
}
```

Obrázek 3.25: K příkladu krokování

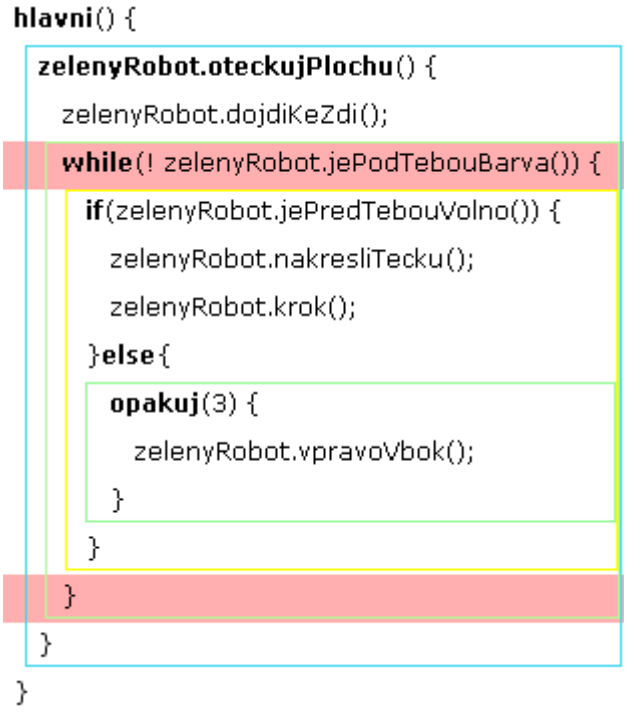
3. Na obrázku 3.26 je situace po druhém stisknutí tlačítka `Krok`. Právě se rozbalila metoda `zelenyRobot.oteckujPlochu()`, tudíž jsou vidět její instrukce a také je vidět, že je bude provádět zelený robot. Červeně je podbarven řádek, na kterém metoda začíná a pro přehlednost i kde končí, což značí, že se bude v dalším kroku provádět tato metoda. Po dalším stisknutí tlačítka se podbarví první řádek v těle metody a při dalším stisknutí se tato instrukce vykoná.
4. Na obrázku 3.27 je vidět, že se jako u metody i u cyklu podbarví začátek a konec. Po stisknutí tlačítka se vyhodnotí podmínka cyklu `while`. Hlava příslušného robota se vybarví podle vyhodnocení podmínky (na obr. 3.23 jsou vidět reakce modrého robota). Vzhledem k negaci podmínky se podbarví první řádek v cyklu a cyklus se bude procházet, pokud podmínka splněna nebude. Pokud bude splněna, podbarví se řádek za blokem cyklu a cyklus se nebude procházet.



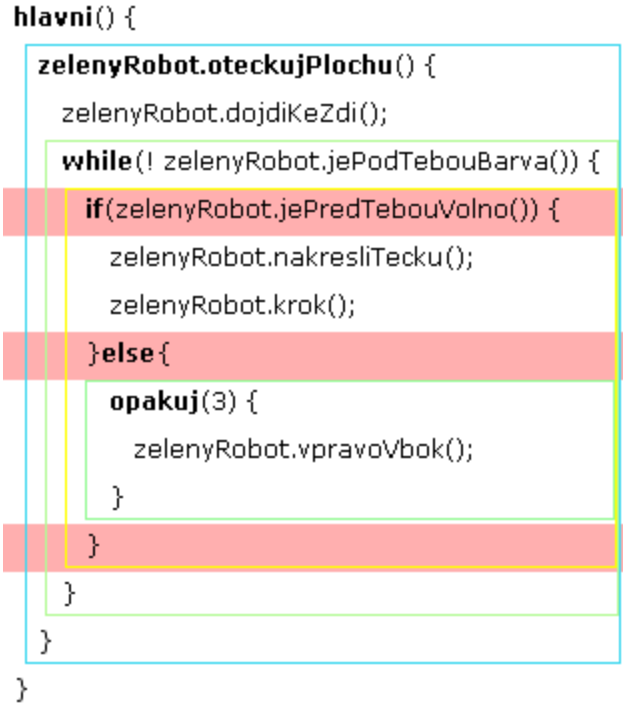
Obrázek 3.26: K příkladu krokování

5. Obrázek 3.28 zobrazuje, že u úplné podmíněné instrukce se při vstupu podbarví tři řádky, které ji tvoří. I zde se při stisknutí tlačítka vyhodnotí podmínka. Pokud bude splněna, vybraví se na animační ploše hlava příslušného robota bíle, nebude-li splněna, vybarví se černě. Pro ilustraci jsou reakce modrého robota vidět na obr. 3.23.
6. Na obrázku 3.29 vidíme rozbalenou metodu `nakresliTecku()`. Po stisknutí tlačítka se provede instrukce `prilozTuzku()`.
7. Obrázek 3.30 ilustruje procházení cyklu `opakuj`. Právě se prochází podruhé ze tří průchodů. Instrukce `zelenyRobot.vpravoVbok()`se provede po stisknutí tlačítka.

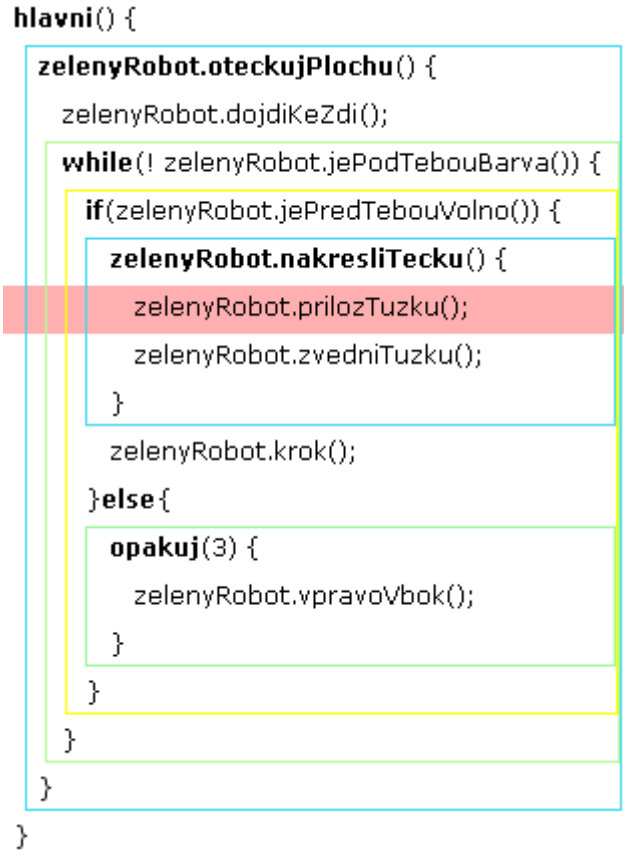
Je zřejmé, že robot v naznačeném algoritmu obkreslí plochu proti směru hodinových ručiček jenom proto, abychom viděli v činnosti cyklus `opakuj`.



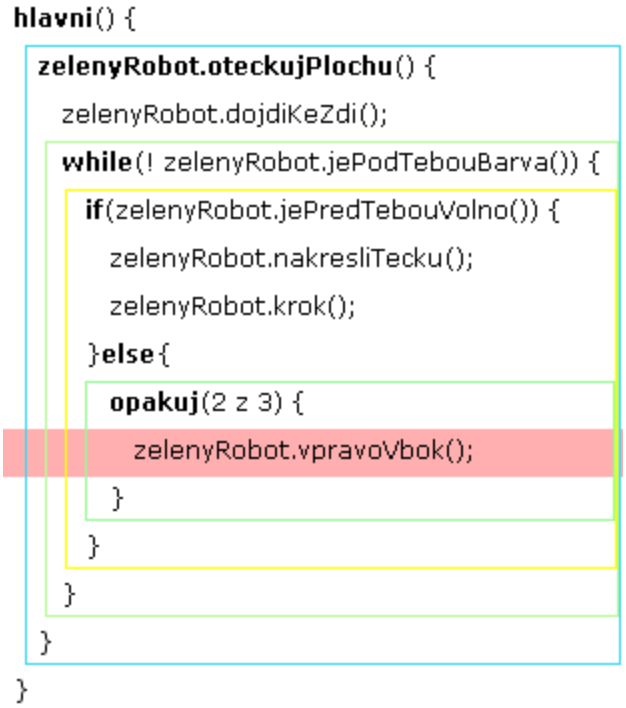
Obrázek 3.27: K příkladu krokování



Obrázek 3.28: K příkladu krokování



Obrázek 3.29: K příkladu krokování



Obrázek 3.30: K příkladu krokování

3.3.10 Ukládání a načítání úloh

Po stisknutí tlačítka `Uložit aktuální úlohu`, druhé tlačítko na panelu nástrojů Panelu animace (obr. 3.3 na straně 50), je možno uložit aktuální konfiguraci, včetně všech vytvořených metod a kódu hlavní metody. Data jsou ukládána do souborů s příponou `.rur`.

Po ukončení běhu aplikace se aktuální úloha uloží automaticky do souboru `last.rur`. Současně s ní se uloží i aktuální nastavení. Soubor se vytvoří, případně přepíše, ve stejném adresáři, kde se nachází právě ukončená aplikace (archiv `RUR.jar`). Při spuštění je soubor naopak automaticky načten, pokud ho tam aplikace nalezne, uložená úloha se zobrazí a vlastnosti nastaví.

Uloženou úlohu lze z datového souboru s příponou `.rur` načíst po stisknutí tlačítka `Načíst úlohu ze souboru`, první tlačítko na panelu nástrojů Panelu animace (obr. 3.3 na straně 50).

Stejným způsobem je možno ze souboru načíst úlohu kurzu.

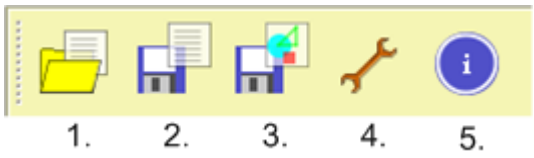
3.3.11 Ukládání obrázků

Máme možnost si přímo uložit aktuální obrázky zobrazované na jednotlivých panelech hlavního okna. Jedná se o obrázek:

- Právě zobrazené animační plochy,
- kódu metody zobrazené na právě aktivní záložce Panelu kódu,
- a nabídky instrukcí na aktivní záložce Panelu nabídky.

Obrázek můžeme uložit po stisknutí příslušného tlačítka odpovídajícího panelu. Obrázek lze uložit ve formátech JPEG a PNG. Pokud není zadaná přípona, obrázek se uloží ve formátu PNG.

3.3.12 Funkce jednotlivých tlačítek

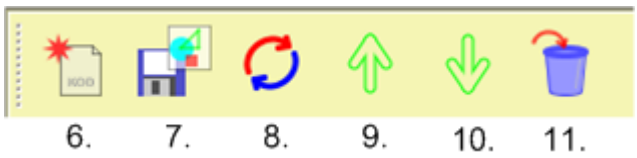


Obrázek 3.31: Tlačítka Panelu animace

Panel animace (obr. 3.31)

1. `Načíst úlohu ze souboru` – umožní načíst ze souboru jinou úlohu kurzu, případně uživatelem již uloženou úlohu, z datového souboru s příponou `.rur`.

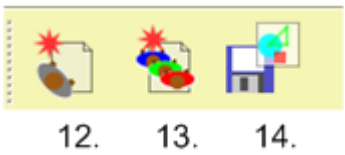
2. `Uložit aktuální úlohu` – umožní uložit aktuální konfiguraci, včetně všech vytvořených metod a kódu hlavní metody. Data jsou ukládána do souborů s příponou `.rur`.
3. `Uložit aktuální obrázek animační plochy` – umožní uložit obrázek právě zobrazené animační plochy na levém panelu okna aplikace. Obrázek lze uložit ve formátech JPEG a PNG. Pokud není zadaná přípona, obrázek se uloží ve formátu PNG.
4. `Nastavení vlastností` – zobrazí se dialogové okno umožňující nastavení velikosti tlačítek v panelech nástrojů aplikace a textu instrukcí kódu metod a instrukcí nabídky.
5. `Info o programu` – zobrazí se informace o vytvoření programu v rámci této diplomové práce.



Obrázek 3.32: Tlačítka Panelu kódu

Panel kódu (obr. 3.32)

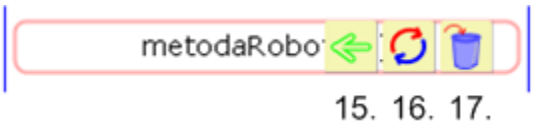
6. `Odstranit všechny instrukce právě zobrazené metody` – obsahuje-li právě zobrazená metoda instrukce a nedošlo od posledního uložení v aplikaci ke změnám, tak se zobrazí zpráva upozorňující na nevratnost této změny, nabídne se možnost uložení úlohy či stornování tohoto požadavku.
7. `Uložit obrázek kódu právě zobrazené metody` – umožní uložit obrázek kódu metody právě zobrazené na prostředním panelu okna aplikace, případně na aktivní záložce. Obrázek lze uložit ve formátech JPEG a PNG. Pokud není zadaná přípona, obrázek se uloží ve formátu PNG.
8. `Upravit` – pokud je před stisknutím tlačítka v kódu právě zobrazené metody vybrána programová konstrukce (cyklus nebo podmíněná instrukce) zobrazí se příslušné dialogové okno umožňující změnu vlastností vybrané konstrukce.
   Pokud byla vybrána vytvořená metoda, zobrazí se na samostatné záložce prostředního panelu, kde je možno ji upravovat.
9. `Posunout o jeden řádek nahoru` – vybraná instrukce kódu (ohraničená růžovým oválem) se posune při stlačení tlačítka vždy o jeden řádek kódu nahoru, vymění se tedy dvě nad sebou zobrazené instrukce.
10. `Posunout o jeden řádek dolu` – narozdíl od předchozího tlačítka se instrukce posune dolů.
11. `Odstranit` – nevratně se z kódu odstraní vybraná instrukce (ohraničená růžovým oválem).



Obrázek 3.33: Tlačítka Panelu nabídky

Panel nabídky (obr. 3.33)

12. `Vytvořit novou metodu robota` – obrazně se vytvoří nová metoda třídy `Robot`, případně třídy `RobotTuzka`. Záleží na tom, který robot má právě zobrazenou nabídku svých instrukcí na aktivní záložce panelu nabídky. Na Panelu kódu se zobrazí záložka této metody a na Panelu nabídky se zobrazí diagram příslušné třídy i s touto novou metodou.
13. `Vytvořit novou obecnou metodu` – narozdíl od metody robota se vytvořená metoda obrazně přidá do třídy `Obecne`.
14. `Uložit obrázek právě zobrazené nabídky` – umožní uložit obrázek aktuální nabídky instrukcí, tedy obrázek zobrazený na pravém panelu okna aplikace, případně na aktivní záložce robota. Obrázek lze uložit ve formátech JPEG a PNG. Pokud není zadaná přípona, obrázek se uloží ve formátu PNG.



Obrázek 3.34: Tlačítka vytvořených metod na Panelu nabídky

Tlačítka vytvořených metod na Panelu nabídky (obr. 3.34)

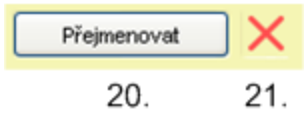
15. `Vložit metodu do kódu` – vybraná metoda nabídky se vloží pod instrukci vybranou v kódu právě zobrazené metody (ohraničená růžovým oválem) na prostředním panelu okna aplikace.
16. `Upravit metodu` – kód vybrané metody se zobrazí na samostatné záložce prostředního panelu okna aplikace. Poté lze metodu upravovat, tedy ji přejmenovat a měnit její kód.
17. `Odstranit metodu z nabídky` – pokud vybraná metoda není vložena do kódu jiné metody, bude nevratně z nabídky odstraněna.

Obrázek 3.35: Tlačítka krokování úlohy

Tlačítka krokování úlohy (obr. 3.35)

18. `Krok` – při prvním stlačení přejde aplikace do krokovacího módu. Kód metod nelze upravovat. Příslušná tlačítka a Panel nabídky se až do stisknutí tlačítka `Stop` zneaktivní. Při dalších stisknutích se postupně krokuje metoda na záložce, která byla aktivní při prvním stlačení tlačítka (hlavní metoda nebo obecná metoda). Při stlačení tlačítka se tedy vykoná aktuální instrukce programu, jejíž řádek je právě červeně podbarven, a příslušné akce se zobrazují na animační ploše. Po ukončení provádění programu se tlačítko až do stlačení tlačítka `Stop` zneaktivní. Tlačítko je neaktivní, pokud je aktivní záložka metody robota.

19. `Stop` – po stlačení tlačítka přejde aplikace z módu krokování programu do módu programování. Provede se restart animační plochy a vlastností robotů, aktivizují se příslušná tlačítka a panely.



Obrázek 3.36: Tlačítka záložky vytvořených metod na Panelu kódu

Tlačítka záložky vytvořené metody na Panelu kódu (obr. 3.36)

20. `Přejmenovat` – po prvním stlačení se zobrazí textové pole, do něhož lze zapsat nový název metody. Po druhém stlačení se ověří vhodnost zadaného názvu. Pokud je název vhodný, tak se metoda přejmenuje a textové pole zmizí. Pokud ne, zobrazí se zpráva upozorňující na nevhodnost názvu.

21. `Uzavřít záložku` – daná záložka zmizí. Metoda se neodstraní, všechny úpravy jsou platné. Metodu lze dále upravovat po novém zobrazení záložky.

Kapitola 4

Možnosti modifikace

Musíme vyjít z předpokladu, že aplikace má sloužit pouze pro úvod do programování. Jak už jsem psal, řada výhod při delším používání aplikace postupně přechází v nevýhody. A je tu samozřejmě hypotetická možnost vytvořit univerzální prostředí, kde z navrhovaného úvodu do výuky programování plynule přejdeme ve skutečné vývojové prostředí, s možností psaní a překladu kódu. Mnohem efektivnější je po úvodu ve vhodný okamžik přejít do jiného prostředí pro tyto účely vyvíjeného.

Zaměřím se tedy na možnosti smysluplné modifikace této aplikace, používané ve smyslu navržených úloh a sloužící tudíž pouze pro úvod do programování. Realizace některých rozšíření by nebylo náročné a nebylo by nutné stávající kód nijak výrazně měnit, zvlášť počítalo-li se s nimi již při návrhu. Ale potřeba poměrně velké části navrhovaných změn a rozšíření aplikace se projevila až při samotném testování a používání programu. Některé z nich by tedy vyžadovaly poměrně velké zásahy do návrhu a rozsáhlý refaktoring kódu. Přesto je tu bez rozdílu uvedu všechny.

- **Editor úloh** – Asi nepřínosnější rozšíření, již od začátku plánované, je WYSIWYG editor. Alespoň pro jednoduchou úpravu již vytvořených úloh.

- **Interaktivní nápověda** – Vytvořit nápovědu spouštěnou přímo z programu. V podstatě by stačilo uživatelskou příručku převézt do html a provázat odkazy.

- **Vytvoření nové metody z části kódu jiné metody** – Asi hlavní nedostatek, který může demotivovat studenty, je nemožnost vybrat několik řádků, které se například v již vytvořeném kódu opakují, a vytvořit z nich novou metodu.

  Tím by se vyřešilo i vytvoření nové metody úpravou již vytvořené metody. Například, když máme metodu umožňující robotu dojít ke zdi, potřebujeme ji rozšířit tak, aby u toho robot ještě dělal tečky a nechceme si starou metodu ničit.

  Navrhované řešení se mi zdá pro vytvoření vhodných programovacích návyků mnohem vhodnější než možnost výběru a kopírování řádků, po kterém volají studenti.

Pokud by se tato úprava neprovedla, tak by se ještě dalo uvažovat o možnosti hromadného posunu vícero vybraných řádků. Například několika instrukcí, které se mají opakovat, přesunout v kódu metody najednou do těla cyklu.

- **Umožnit nevykonávání v kódu označených instrukcí** – Tzv. vyremování – vytvoření poznámky.

  Případně umožnit přímo vytvářet poznámky. Ale jsem převědčen, že když jsou studenti při dekompozici úlohy vedeni k vytváření a vhodnému pojmenování metod, tak poznámky nejsou potřeba. I bez nich se dá vytvořit hezký, čitelný kód a u takto jednoduchých úloh není dokumentace ještě nutná.

- **Sjednotit ovládání** – Mám tím namysli možnost používat při úpravě instrukcí v kódu metody obdobná tlačítka, jako při úpravě metod v nabídce (obr. 3.34). Je to sice netradiční, ale při určitých úpravách je to efektivnější a člověk si na to snadno zvykne. Momentální tlačítka panelu kódu by samozřejmě zůstala beze změny.

- **Akce zpět a znovu** – Alespoň o jednu akci. U některých nevratných akcí jsme sice dotazováni a upozorněni na možnost uložení, ale obecně to spíše obtěžuje a mnohdy si až po provedení úprav plně uvědomíme, co vše se stalo.

- **Nastavení barev prostředí** – Téměř bez problémů by šla měnit barva pozadí oken, barva písma, barvy objektů použitých na animační ploše. Případně vybírat z nekolika barevných módů.

- **Další objekty a dovednosti robotů** – Zde se nabízí velký prostor pro fantazii. Jen je nutno myslet na zaměření této aplikace – úvod do programování. Jako příklad uvádím dva objekty.

  - **Guma** – Robot by mohl tužku otočit a případně gumovat jen svou či jakoukoliv barvu. Nebo by mohl tužku odložit a vzít si gumu, jakožto samostatný objekt.

    Velmi zajímavou možností je tužka, která si při opětovném kreslení sama maže. Zkoušeli jsme to se studenty při programování na papír a umožňuje to řešení poměrně zajímavých problémů.

  - **Časovaná bomba** – Robot by mohl zvedat a přenášet bombu, která by po aktivování po nastaveném počtu kroků explodovala. V úlohách se tím nabízí možnost interakce robotů a hledání efektivnějších algoritmů.

- **Logické spojky v podmínkách** – Možnost vkládat vícero podmínek a provázat je pomocí logických spojek AND a OR.

- **Celočíselné parametry metod** – Používání celočíselných proměných jakožto parametru metody. Například hranu čtverce, kterou by bylo možné využít při určení počtu opakování.

  S tím souvisí i možnost použití vzorce se základními celočíselnými operacemi při určování počtu opakování, včetně celočíselného dělení a zbytku po dělení. Například k určování poloviny strany.

- **Ozvučení akcí** – Start krokování (prvotní stisknutí tlačítka `Krok`), krok, výbuch, ukončení krokování, stisknutí tlačítka `Stop`.

- **Menu** – při realizaci některých návrhů by bylo již vhodné vytvořit klasické menu aplikace.

- **Změna zobrazování textového okna přejmenování metody** – V nastavení vlastností aplikace by bylo možné předvolit, zda má být po vytvoření metody textové okno přejmenování metody aktivně připravené k přejmenování.

  Studeni jsou cíleně vedeni k vhodnému pojmenovávání metod a takto by k tomu byli přirozeně vyzýváni. Avšak při intuitivním seznamování s prostředím je naopak vhodnější, aby prostředí bylo přehlednější.

- **Instalace programu** – Java je multiplatformní, ale přesto by alespoň pro operační systémy Windows bylo vhodné umožnit automatickou instalaci. Nejspíš by se použil některý z volně šiřitelných instalačních programů.

  Instalátor by vložil potřebné klíče do registů, umístil hlavní program `RUR.jar` nejsp. do složky `Program Files`, v menu by přibyla položka a na ploše ikonka umožňující spuštění programu s poslední řešenou úlohou.

  Hlavní přínos by byl však v možnosti přímého spouštění dílčích úloh (s příponou `.rur`), což bylo vyzkoušeno a je to možné po vložení potřebných klíčů do registrů již teď.

- **Zneaktivnění pro akci nepoužitelných tlačítek** – Momentálně se tato tlačítka nijak neodlišují, např. nelze-li upravovat vybranou instrukci (třaba krok robota), nebo ji nelze posunout ještě více v kódu, atd. Hlavním důvodem je momentální řešení rozlišení dvou módů aplikace. Jednak módu vytváření kódu a jednak módu krokování programu, ve kterém jsou ikonky všech nepřípustných tlačítek šedivá.

  S tím tedy samozřejmě souvisí potřeba navržení a realizace jiného, dalšího vizuálního rozlišení těchto dvou módů aplikace.

- **Applety** – Nabízí se také možnost přenést úlohy na internet. Vytvořit tak on-line kurz využívající appletů.

Kapitola 5

Závěr

V rámci této diplomové práce byla vytvořena aplikace umožňující úvod do výuky programování na středních školách, případně na VŠ. K aplikaci je připravena, a v diplomové práci přehledně popsána, sada úloh obsahující 46 úkolů s metodickými pokyny pro jejich využití ve výuce. Jako vyšší programovací jazyk byl zvolen moderní a rozšířený objektově orientovaný jazyk JAVA. Součástí diplomové práce je také uživatelská příručka obsahující přehledné návody pro práci s aplikací.

Všechny základní soubory aplikace jsou sbaleny do jednoho archivu – `RUR.jar`. Jednotlivé úlohy jsou v datových souborech s příponou `.rur`. Výhodou je, že soubor `RUR.jar` lze spouštět na jakékoliv platformě, pro kterou máme k dispozici příslušný virtuální stroj jazyka Java JVM.

Sada navržených úloh není učebnicí, je pouze metodickou pomůckou pro učitele, který ji může využívat při své přípravě vyučovací hodiny a stanovení konkrétních cílů. Kurz vytvořený učitelem se pak pro studenty může stát snadněji pochopitelným a ne tak náročným jako většina klasických kurzů programování.

Návrh úloh vychází z cílů stanovených rámcovým vzdělávacím programem pro obor vzdělávání Technické lyceum. Nepokrývá veškeré stanovené cíle RVP, slouží pouze pro úvod do výuky. Studenti se s novými pojmy a principy seznamují v jednotlivých úlohách formou hry. Programují chování robota, přičemž jakákoliv akce je v prostředí viditelná, čímž se snižuje stupeň abstrakce. Studenti nejsou zahlcováni informacemi, naopak se snaží s poměrně omezenými možnostmi vyřešit přiměřeně náročný a jasně definovaný problém. Zažívají úspěch a jsou přirozeně motivováni k řešení dalších úloh a postupnému osvojování nových pojmů, konstrukcí a principů, které jim umožní daný problém úspěšně vyřešit. Bohatost jazyka postupně vnímají jako výhodu.

Vytvořená aplikace a dílčí úlohy bohužel nestačily projít komplexnějším testováním, jedná se tedy o tzv. beta verzi. Domnívám se však, že mnou vytvořená aplikace je zdařilá, prostředí je přehledné, s intuitivním ovládáním i na dotykové tabuli. Některé úlohy byly studenty vyzkoušeny během výuky povinně volitelného předmětu – programování na SPŠ stavební v Plzni. Myslím si, že je práce bavila a stanovené cíle byly úspěšně naplňovány. Projevené nedostatky, další nápady a náměty jsem popsal v kapitole Výhody a nevýhody. Úpravy či rozšíření aplikace jsem navrhl

v kapitole Možnosti modifikace. Použití aplikace pro úvod do programování mě přineslo převážně pozitivní zkušenosti. Z negativních zkušeností uvedu dvě podstatné. To, že studenti nepíší kód, považuji v úvodu výuky za výhodu, ale ukázalo se, že pro studenty je snazší tzv. naklikat rychle několik stejných instrukcí za sebou, než nastavovat počet opakování. Asi hlavní nedostatek aplikace, který se projevil při jejím používání a demotivoval studenty, byl v nemožnosti vybrat několik řádků kódu a vytvořit z nich novou metodu.

Domnívám se, že aplikaci s úlohami je vhodné zařadit na úvod do výuky programování na SŠ, zvláště po provedení navrhovaných změn a doladění případných chybiček objevených při komplexnějším testování.

Seznam obrázků

Literatura

[1] ECKEL, Bruce. *Myslíme v jazyku Java: knihovna programátora.* Kiszka Bogdan. Praha: Grada Publishing, 2001. 432 s. ISBN 80-247-9010-6.

[2] GOSLING, James et al.. *The Java Language Specification, Second Edition* [online]. Sun Microsystems, 2000 [cit. 2009-03-10].
Dostupný z WWW:
<http://java.sun.com/docs/books/jls/second_edition/html/j.title.doc.html>.

[3] HAWLITZEK, Florian. *Java 2: příručka programátora.* Bráza Jiří. Praha: Grada Publishing, 2002. 316 s. ISBN 80-247-9060-2.

[4] SCHMULLER, Joseph. *Myslíme v jazyku UML: knihovna programátora.* Praha: Grada Publishing, 2001. 360 s. ISBN 80-247-0029-8.

[5] PECINOVSKÝ, Rudolf. *Myslíme objektově v jazyku Java 5.0.* Praha: Grada Publishing, 2004. 604 s. ISBN 80-247-0941-4.

[6] *Rámcový vzdělávací program pro obor vzdělání 78-42-M/01 Technické lyceum*[online]. MŠMT, 2007 [cit. 2009-03-10]. Dostupný z WWW:
<http://zpd.nuov.cz/RVP/ML/RVP%207842M01%20Technicke%20lyceum.pdf>.

[7] VIRIUS, Miroslav. *Java pro zelenáče.* Praha: Neocortex, 2001. 240 s. ISBN 80-902230-9-5.

[8] VIRIUS, Miroslav. *Základy algoritmizace.* 2. přepracované vydání. Praha: Česká technika – nakladatelství ČVUT, 2008. 265 s. ISBN 978-80-01-04003-4.

Kapitola 6

Summary

Within the dissertation it was created an application which clears the way for an introduction into programming teaching at secondary schools, alternatively at universities. For the application there is prepared a set of tasks with methodical instructions for their use in teaching. The draft of tasks proceeds from the aims which are determined by a framework educational programme for „Technical lycee“. It doesn't cover all the determined aims, it serves only as an introduction into teaching. As a higher programming languauge was selected modern and widely used language JAVA.

The set of tasks isn't a textbook, it's only a methodical aid for teachers who can prepare lessons and determine specific aims with its help. The course made by a teacher can students understand more easily and it isn't so exacting as most classic programming courses.

Students acquaint themselves with new terms and principles in individual tasks in a form of a game. They programme behaviour of a robot and so any operation is visible which reduces degree of abstraction. Students aren't overloaded with information. On the contrary students try to solve adequately difficult and clearly defined problem with relatively restricted possibilities. They experience success and they are naturaly motivated to solution to other tasks and gradual learning new terms, constructions and principles which enable them to solve the problem successfully. They gradually perceive richness of language as an advantage.

The application and set of tasks is suggested only as an introduction into programming because lots of advantages are gradually changing into disadvatages and students can gain improper habits. In fact they won't learn programming in general.

Teaching with the help of this application is focused on solutions of problem when making algorithms, on understanding of principles and connections. It teaches students to think.

A part of the dissertation is also a user's manual containing well arranged instructions for work with the application. In the work there are also analyse advantages and disadvantages of the use of this application and posibility of its modification.