UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

SYSTÉM NA GENEROVANIE ÚLOH NA VYUČOVANIE ZÁKLADOV PROGRAMOVANIA

Bakalárska práca

2014 **Lucia Budinská**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

SYSTÉM NA GENEROVANIE ÚLOH NA VYUČOVANIE ZÁKLADOV PROGRAMOVANIA

Bakalárska práca

Študijný program:	Aplikovaná informatika
Študijný odbor:	2511 Aplikovaná informatika
Školiace pracovisko:	Katedra aplikovanej informatiky
Školiteľ:	doc. PaedDr. Monika Tomcsányiová, PhD.
Konzultant:	RNDr. Peter Borovanský, PhD.

Bratislava, 2014 **Lucia Budinská**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Lucia Budinská
Študijný program: aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Systém na generovanie úloh na vyučovanie základov programovania / *A system for generating tasks for teaching programming basics*

Cieľ:
- zistiť, aké typy programátorských úloh sú vhodné pre žiakov na 1. stupni ZŠ
- navrhnúť systém, v ktorom bude možné pripraviť základné šablóny pre vyšpecifikované typy úloh na vyučovanie základov programovania na 1. stupni ZŠ
- doplniť do systému vlastnosti, ktoré pomôžu učiteľovi vygenerovať vhodné úlohy pre žiakov podľa zadaných kritérií
- navrhnúť spôsob prezentácie úloh žiakom a tiež spôsob ich vyhodnocovani

Vedúci: doc. PaedDr. Monika Tomcsányiová, PhD.
Konzultant: RNDr. Peter Borovanský, PhD.
Katedra: FMFI.KZVI - Katedra základov a vyučovania informatiky
Vedúci katedry: doc. RNDr. Zuzana Kubincová, PhD.
prof. RNDr. Ivan Kalaš, CSc.

Dátum zadania: 22.10.2013

Dátum schválenia: 28.10.2013

doc. RNDr. Damas Gruska, PhD.
garant študijného programu

študent

vedúci práce,
konzultant

Čestné vyhlásenie

Čestne vyhlasujem, že som bakalársku prácu „Systém na generovanie úloh na vyučovanie základov programovania" vypracovala samostatne s použitím uvedenej literatúry a zdrojov dostupných na internete.

V Bratislave dňa 30.5.2014 _____

Lucia Budinská

Poďakovanie

Rada by som sa poďakovala doc. PaedDr. Monike Tomcsányiovej za odborné vedenie, čas, ktorý mne a tejto práci venovala, podporu a veľmi cenné rady a návrhy, ktoré si naozaj vážim. Taktiež sa chcem poďakovať RNDr. Petrovi Borovanskému, PhD. za jeho odporúčania a nápady, ktoré pomohli zlepšiť túto prácu.

Abstrakt

BUDINSKÁ, Lucia: *Systém na generovanie úloh na vyučovanie základov programovania (Bakalárska práca)* – Univerzita Komenského v Bratislave, Fakulta matematiky, fyziky a informatiky; Katedra aplikovanej informatiky. – Školiteľ: doc. PaedDr. Monika Tomcsányiová, PhD., Konzultant: RNDr. Peter Borovanský, PhD.: FMFI UK, 2014, 40 strán

Cieľom bakalárskej práce je preskúmať dostupné programovacie prostredia používané na prvom stupni základných škôl, zadefinovať najčastejšie používané typy úloh a na základe zistených poznatkov navrhnúť a vytvoriť systém, ktorý umožní učiteľom vytvárať podobné úlohy a zadávať ich svojim žiakom. Systém obsahuje vlastnosti, vďaka ktorým je možné generovať úlohy zo vstupných parametrov od učiteľa. Vygenerované úlohy môžu žiaci priamo v programe riešiť a systém dokáže vyhodnocovať ich riešenia. Ich výsledky sú exportované do tabuľky pre učiteľa. Vytvorenie aplikácie v jazyku Java umožňuje nasadenie aplikácie lokálne na počítače používateľa, ale aj globálne na internete.

Kľúčové slová: základy programovania, generovanie, úlohy, vyhodnocovanie, programovacie prostredie

Abstract

BUDINSKÁ, Lucia: *A system for generating tasks for teaching programming basics (Bachelor thesis)* – Comenius University in Bratislava, Faculty of Mathematics, Physics and Informatics; Department of Applied Informatics – Supervisor: doc. PaedDr. Monika Tomcsányiová, PhD., Consultant: RNDr. Peter Borovanský, PhD.: FMFI UK, 2014, 40 pages

The main goal of the bachelor thesis is to study existing programming environments used in primary schools, define the most used types of tasks and from the determined results design and create a system which allows teachers to create similar types of tasks and assign them to their pupils. The system includes functions which helps the teacher with generating tasks from his input parameters. Pupils can solve generated tasks directly in the program and the system evaluates their solutions. Their results can be exported into the sheet for the teacher. Application is created in Java programming language therefore it can be used either locally on user's computer or globally on the internet.

Keywords: programming basics, generating, tasks, evaluation, programming environment

Predhovor

Informatika sa stala súčasťou života každého z nás. Pre najmladších školopovinných žiakov je povinná v rámci predmetu informatická výchova na základných školách. Na prvý stupeň sa tak presunulo aj vyučovanie základov programovania.

Existuje viacero programovacích prostredí, ktoré učitelia na vyučovanie používajú, ale tie im často neponúkajú veľa možností, ako zjednodušiť prípravu zadaní alebo pomôcť s vyhodnocovaním riešení. Taktiež je každé prostredie orientované na iný typ úloh, čo prináša problémy s prechodmi medzi jednotlivými prostrediami, alebo iba jednosmerné orientovanie výučby.

Pre žiakov je dôležité, aby pochopili základy programovania. Práve z toho dôvodu musíme žiakom ponúkať rozmanité zadania, ktoré ich nútia rozmýšľať a vyučovanie programovania sa tak nestane zautomatizovanou činnosťou.

Vytvorenie systému, ktoré je cieľom tejto práce, má pomôcť učiteľom aj žiakom zjednodušiť a spestriť výučbu.

Obsah

Úvod

Žijeme v dobe, keď sú počítače pre žiakov na základných školách už samozrejmosťou a väčšina z nich nemá problémy s ich ovládaním. Preto tomu musíme prispôsobiť aj vyučovanie, zmodernizovať ho a zatraktívniť. Jednou z možností je vyučovať hravou formou, zamerať sa na konkrétnych žiakov a čo najviac im prispôsobiť danú tému.

Keďže každá skupina žiakov je iná, je dôležité, aby učiteľ mohol pre každú z nich ľahko a rýchlo vytvárať úlohy, resp. sady úloh, či už podľa ich vedomostí alebo nejakej konkrétnej témy. Aby bolo vytváranie úloh pre učiteľa časovo nenáročné, systém by mal generovať úlohy na základe niekoľko málo vstupných parametrov a pre rovnaké parametre ponúknuť viacero úloh, z ktorých si učiteľ môže vybrať.

Vo väčších skupinách sa nemôže učiteľ venovať každému žiakovi individuálne, preto je dôležité, aby programovací jazyk, resp. používané vývojové prostredie, so žiakom komunikovalo, ukázalo, kde má nedostatky a pomohlo mu ich odstrániť. Zároveň by však výsledky každého žiaka mali byť učiteľovi k dispozícii, aby mal z hodiny spätnú väzbu a vedel, ako jeho žiaci rozumejú danej téme.

Naše prostredie ponúkne práve takýto komplexný nástroj, ktorý zjednoduší učiteľovu prácu a zároveň žiakom poskytne prostriedok na rozvíjanie svojich poznatkov aj mimo školy.

V úvodnej kapitole sa zameriame na existujúce prostredia a úlohy, využívané na vyučovanie základov programovania na 1. stupni ZŠ. Taktiež sa sústredíme na teoretické poznatky potrebné na tvorbu edukačného softvéru. V druhej kapitole, na základe zistených poznatkov, navrhneme, ako by mal takýto systém vyzerať, aké funkcie má ponúkať a aké vlastnosti by mal obsahovať. Záverečná kapitola bude obsahovať popis nami vytvoreného systému, jednotlivé časti implementácie a pohľad na obe používateľské rozhrania.

1. Analýza programovacích prostredí

V úvodnej kapitole bližšie opíšeme dostupné programovacie prostredia a jazyky pre žiakov na 1. stupni ZŠ a zanalyzujeme dostupné aktivity na rozvoj algoritmického myslenia. Zároveň uvedieme niekoľko informácií dôležitých pri tvorbe softvéru pre túto vekovú skupinu a základné podmienky, ktoré má spĺňať im určený programovací jazyk.

1.1. Programovacie jazyky pre deti

Pred pár rokmi sa začalo vyučovanie informatiky na prvom stupni základných škôl. Základy programovania sa tak začali postupne posúvať k mladšej vekovej kategórii, pre ktorú nie je ešte vhodné programovanie vo vyšších programovacích jazykoch ako je napríklad C++, Java alebo Python. Na učebné účely pre týchto žiakov boli preto vyvinuté takzvané mini jazyky – programovacie jazyky určené pre deti.

Podľa [1] je základnou charakteristikou mini jazyka:

- jednoduchosť v syntaxi aj sémantike,
- názornosť,
- atraktívnosť a zmysluplnosť,
- dialógový režim,
- modularita.

Prostredie určené na výučbu programovania u detí by preto malo byť čo najjednoduchšie, najlepšie bez nutnosti sa učiť syntax (napr. obrázkové príkazy namiesto textových) a všetky operácie by mali byť hneď viditeľné na obrazovke (napr. vykonávanie príkazov postupne, nie naraz). Zároveň by malo prostredie ponúkať témy, ktoré sú pre žiakov zaujímavé. Softvér by mal so žiakom komunikovať a poskytovať mu možnosť vytvárať vlastné, neskôr použiteľné procedúry.

Z množstva mini programovacích jazykov vyberáme niektoré, ktoré sa bežne používajú na slovenských základných školách.

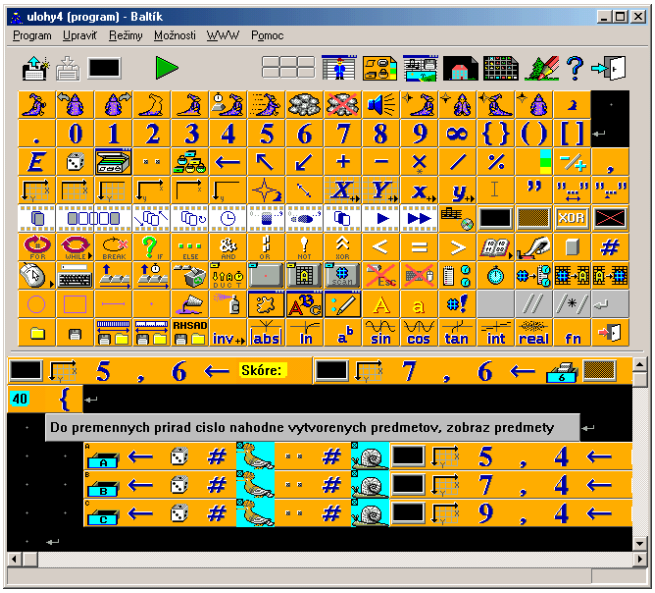
1.1.1. Baltík

Baltík, pozri [2], je programovací jazyk zameraný na vyučovanie základov programovania. Obsahuje niekoľko rôznych úrovní, preto je použiteľný pre viac vekových kategórií.

Všetky príkazy sú zadávané pomocou ikoniek a každá chyba je hneď zvýraznená, čo uľahčuje detskému používateľovi zamerať sa na obsah a algoritmickú stránku programu a nie na syntax jazyka.

Prostredie ponúka štyri pracovné režimy – Skladať scénu, Čarovať scénu, Programovať – začiatočník a Programovať – pokročilý. V prvom režime sa neprogramuje, v druhom sa žiaci zoznamujú so základnými princípmi postupnosti príkazov, zvyšné dva sú určené na priame programovanie podľa úrovne užívateľa. Pre deti na prvom stupni ZŠ je vhodný prvý až tretí režim.

Prostredie Baltík je určené hlavne na vytváranie samostatných úloh a podporu tvorivosti žiakov, preto v ňom nie je kontrola algoritmickej správnosti riešenia. V programe absentuje aj učiteľské prostredie na vytváranie úloh, ktoré by žiaci následne riešili. Existuje však niekoľko zbierok príkladov, ktoré môže učiteľ využiť pri vyučovaní programovania v tomto prostredí.



Obr. 1: Programovacie prostredie Baltík

1.1.2. Robot Karel – Robot Emil 3D

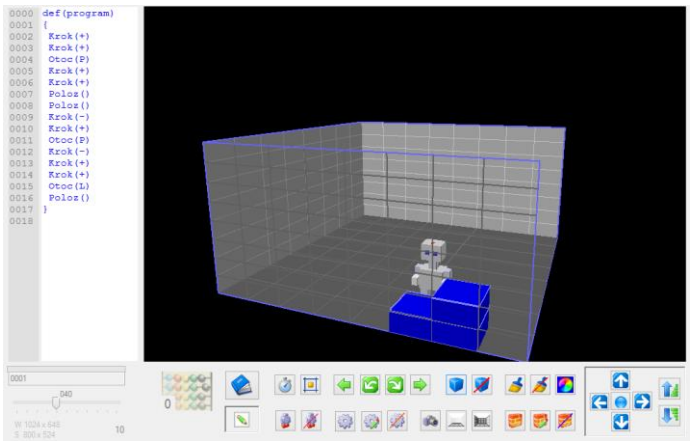
Robot Karel, pozri [3], je programovací jazyk a vývojové prostredie určené na vyučovanie základov objektovo orientovaného programovania. Základnou myšlienkou práce v prostredí je pomocou jednoduchých príkazov posúvať robota po štvorčekovej hracej ploche, resp. ukladať na túto plochu rôzne tehličky.

Na Slovensku vznikla v 90-tych rokoch vylepšená verzia Karel 3D, v ktorej sa robot pohybuje po trojrozmernom bludisku. V roku 2008 bola táto verzia implementovaná do prostredia Robot Emil 3D, pozri [4]. Toto prostredie poskytuje väčšiu škálu možností a zvyšuje tak jeho atraktívnosť pre žiakov.

V prostredí Robot Emil 3D sa robot pohybuje v trojrozmernej miestnosti, na ktorú môžeme pozerať z rôznych uhlov, a vie stavať tehličky rôznych farieb a tvarov, čo zlepšuje priestorové videnie žiakov. Prostredie umožňuje vytváranie vlastných podmienok, čím sa približuje vyšším programovacím jazykom. Robot sa ovláda príkazmi zadávanými textovou formou, alebo ikonkami šípok, ktoré priamo vykonávajú príkazy. Príkazy sa zapisujú aj do textového okna – žiak tak vidí aj syntax jazyka. Tá je jednoduchá na pochopenie – obsahuje prirodzené slová zo slovenského jazyka (resp. jazykovej mutácie).

Úlohy, ktoré možno v tomto jazyku riešiť, sú zamerané práve na 3D priestor a ukladanie kociek, takže neponúka veľké množstvo rozmanitých problémov. Práve kvôli pohybu v priestore, keďže priestorová predstavivosť spolu s relatívnym pohybom robia žiakom všeobecne problémy, je vhodnejší pre starších žiakov.

V jazyku rovnako chýba kontrola správnosti riešenia a prostredie určené na tvorbu zadaní.



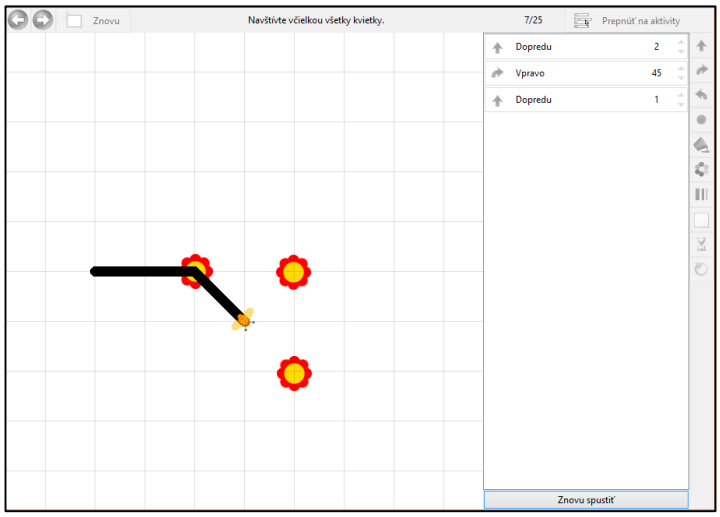
Obr. 2: Programovacie prostredie Robot Emil 3D

1.1.3. EasyLogo

EasyLogo, pozri [5], je programovací jazyk založený na princípoch programovacieho jazyka Logo. EasyLogo je však zjednodušené na použitie pre mladších žiakov a úplných začiatočníkov. Hlavných rozdielom oproti Logu je programovanie pomocou kartičiek – žiak tak nepíše priamo kód, ale ukladá kartičky a tak navrhuje svoj program. Základom tohto

prostredia je štvorčeková mriežka v ktorej sa pohybuje postavička, ktorá sa vie otáčať o 45° a kresliť po ploche obrazce.

Jazyk je zjednodušený – obsahuje iba kreslenie, cykly a procedúry. Prostredie ponúka niekoľko aktivít slúžiacich na naučenie základných programovacích a algoritmických zručností. Každý krok prebehne hneď po zadaní inštrukcie, žiak vie však ešte aj ďalej upravovať svoje riešenie. Ani prostredie EasyLogo nekontroluje zadané riešenie. Poskytuje však programovací mód, v ktorom si žiaci vedia vytvárať vlastné programy a zároveň nové aktivity vie vytvárať aj učiteľ. Tie však musí pripravovať v textovom súbore, pretože program neponúka vytváranie aktivít priamo v prostredí.



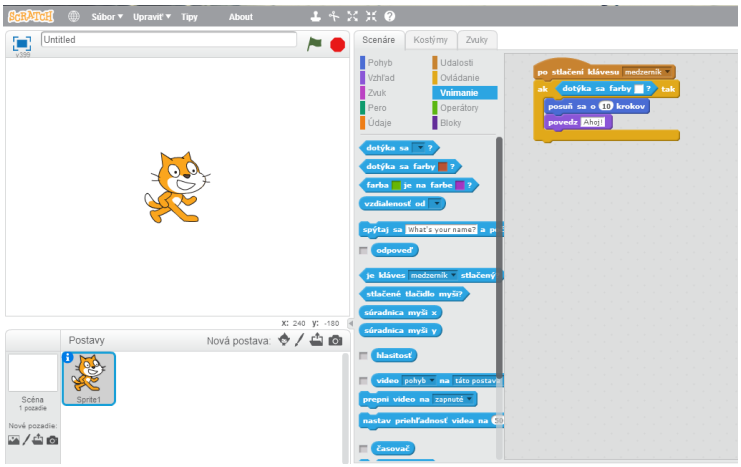
Obr. 3: Programovacie prostredie EasyLogo

1.1.4. Scratch

Scratch, pozri [6], je programovací jazyk určený na výučbu programovania. Programy sa nepíšu priamo, ale skladajú sa pomocou špeciálne tvarovaných kartičiek pripomínajúcich puzzle. Prostredie je intuitívne, tvary kartičiek zabraňujú syntaktickým chybám (dve nesúvisiace kartičky sa nedajú spojiť). Prostredie ponúka veľký výber jednotlivých funkcií a umožňuje žiakom vytvárať multimediálne aplikácie a prezentácie a zložitejšie programy, je preto vhodné aj na použitie vo vyšších ročníkoch.

Pre žiakov v prvých ročníkoch základnej školy však môže byť problematické práve veľké množstvo rôznych príkazov, ktorým ešte nemusia úplne rozumieť, a zároveň textové informácie na kartičkách.

V prostredí nie je zabudovaná kontrola riešenia, keďže Scratch priamo neponúka predpripravené úlohy. Všetky projekty vie žiak ukladať a zdieľať na internete. Zároveň tak vie pripraviť zadanie aj učiteľ – vo forme nejakého rozpracovaného projektu, ktorý bude zdieľať so svojimi žiakmi a tí ho môžu ďalej upravovať.



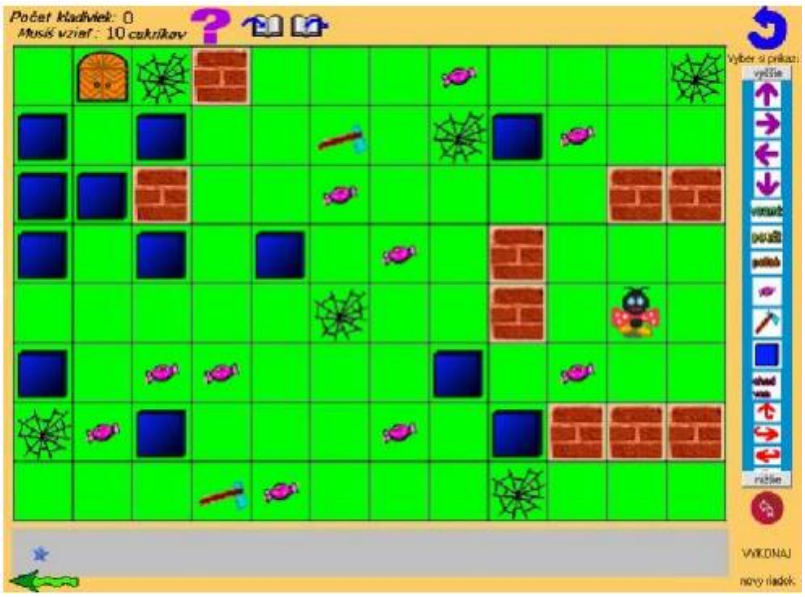
Obr. 4: Programovacie prostredie Scratch

1.1.5. Mravec

Mravec je detský programovací jazyk, ktorý vznikol ako diplomová práca Jany Ondkovej na FMFI, pozri [7]. Žiaci v prostredí štvorčekovej siete pracujú s postavičkou mravca, ktorého môžu ovládať v troch režimoch – priamy (pomocou šípok na klávesnici), šípkový (mravec sa hýbe sám, jeho smer sa mení pomocou šípok uložených do plochy) a ikonkový režim (ovládanie pomocou ikoniek, ktoré predstavujú programovacie príkazy). Mravec sa pohybuje po štvorcovej ploche s rôznymi predmetmi a úlohou je dostať ho ku dverám.

Program kontroluje správnosť riešenia a v prostredí existuje viacero úrovní náročnosti, ktoré sú vhodné na priebežné učenie programovania.

Toto prostredie ponúka aj učiteľský mód na vytváranie úloh pre žiakov podľa náročnosti. Takýmto spôsobom si aj žiaci dokážu vytvárať vlastné úlohy.



Obr. 5: Programovacie prostredie Mravec

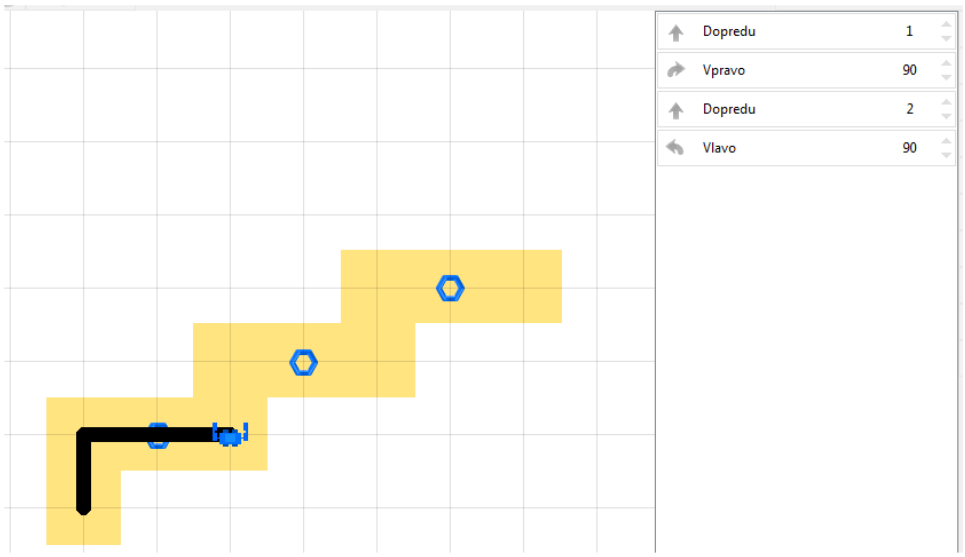
1.2. Programovacie aktivity pre žiakov

V tejto podkapitole sa bližšie pozrieme na niekoľko typov aktivít podporujúcich algoritmické myslenie a takých, ktoré učia žiakov základy programovania. Kategórie sme vytvorili na základe porovnania príkladov z učebníc programovania pre deti, príkladov z vyššie spomenutých mini jazykov a príkladov z informatickej súťaže iBobor.

1.2.1. Pohyb postavičky v štvorčekovej ploche

Typický príklad väčšiny malých programovacích jazykov je dopraviť v štvorčekovej ploche pomocou príkazov postavičku na nejaké miesto. Pohyb je riadený príkazmi ako sú krok dopredu, otoč sa vpravo, otoč sa vľavo (pri absolútnom pohybe dopredu, dozadu, vpravo, vľavo) a cieľom aktivity je naučiť žiakov, ako sa postavička ovláda a čo znamená postupnosť krokov.

Vzorový príklad je z prostredia Easy Logo, pozri [5]. Úlohou je viesť robota po vyznačenej trase s použitím troch príkazových kartičiek – dopredu, vpravo, vľavo.



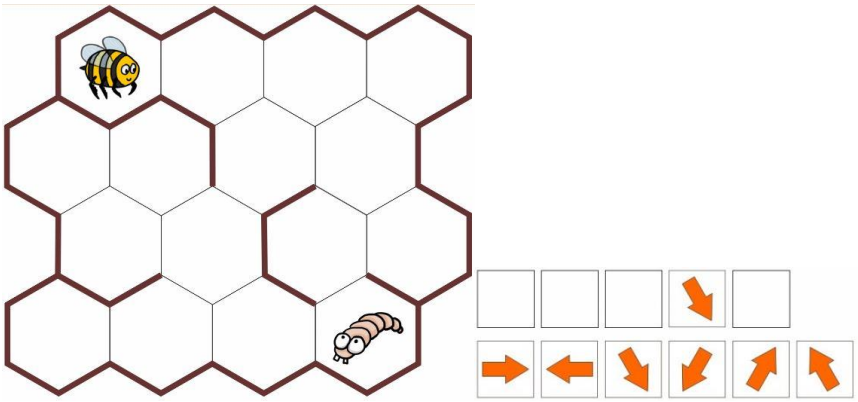
Obr. 6: Úloha z prostredia EasyLogo

1.2.2. Hľadanie cesty v bludisku

Ďalším zo série príkladov, ktoré sa využívajú na vyučovanie základov programovania, je hľadanie cesty v bludisku. Zmenou oproti prvému spomenutému typu úloh je nehomogénnosť pracovnej plochy – t. j. nachádzajú sa tam steny, cez ktoré postavička neprejde, sú tam slepé uličky atď. Úloha je náročnejšia na logické myslenie, z programátorskej stránky precvičuje syntax jazyka.

Rozšírením tohto typu úloh môže byť hľadanie najkratšej cesty.

Ukážkový príklad na obr. 7 je zo súťaže iBobor, pozri [8], príklad bol zadaný v školskom roku 2013/2014 v súťažnej kategórii Bobríci (určenej pre 3.-4. ročník ZŠ). Úlohou bolo pomocou postupností šípok dopraviť včielku k larve. Predposledná šípka bola zadaná.

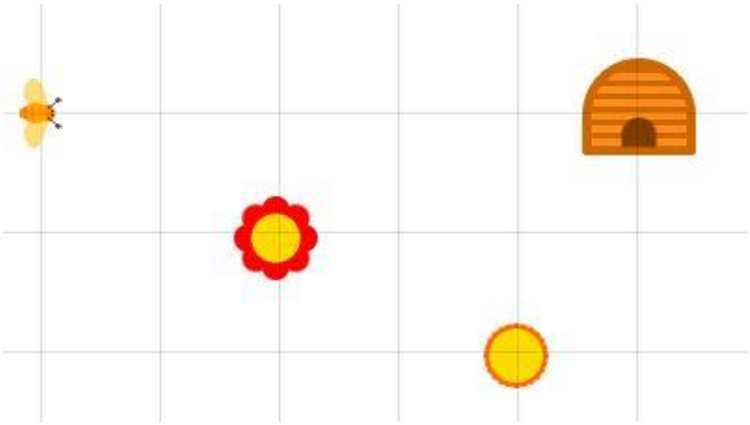


Obr. 7: Úloha zo súťaže iBobor

1.2.3. Zbieranie predmetov

Základnou myšlienkou tohto typu príkladu je prejsť štvorčekovou plochou a pozbierať všetky predmety (buď len prejsť po políčku, na ktorom ležia, alebo ich špeciálnym príkazom vziať). Pomocou nich má žiak vytvorenú imaginárnu cestu, ktorou má ísť. V tomto type úloh sa môžu využívať podmienky (napr. ak je na políčku hračka, vezmi ju) alebo cykly, ak sú predmety usporiadané pravidelne.

Ukážkový príklad je z prostredia Easy Logo, pozri [5]. Úlohou je navštíviť všetky kvietky a dopraviť včeličku do úľa.



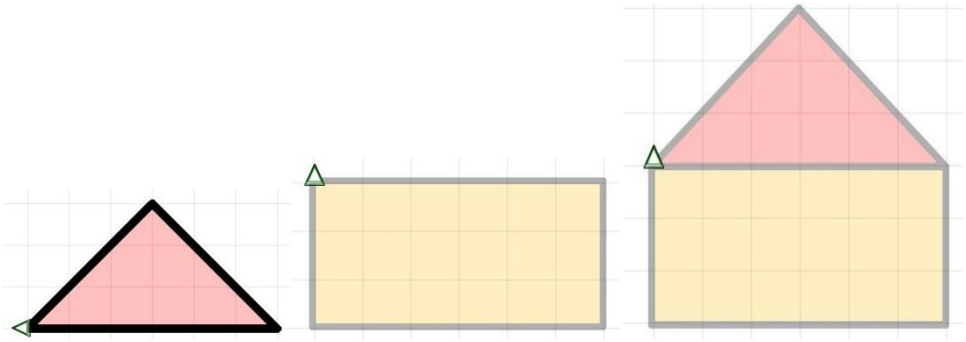
Obr. 8: Úloha z prostredia EasyLogo

1.2.4. Kreslenie tvarov

Najrozšírenejšou úlohou v prostrediach založených na korytnačej grafike je práve kreslenie v štvorčekovej ploche. Žiaci pomocou príkazov pohybujú postavičkou alebo perom po ploche a kreslia základné geometrické obrazce. Zo začiatku jednoduchšie (napr. štvorec, trojuholník, kruh...), neskôr spájajú jednoduchšie tvary do zložitejších (napr. dom).

Pri kreslení sa využíva konštrukcia jednoduchých cyklov, neskôr aj vytváranie vlastných podprocedúr a ich volanie v hlavnej procedúre.

Príklady sú z prostredia Easy Logo, kde sa žiak najprv naučí kresliť trojuholník, potom obdĺžnik a nakoniec sa naučí nakresliť spojením týchto procedúr dom.

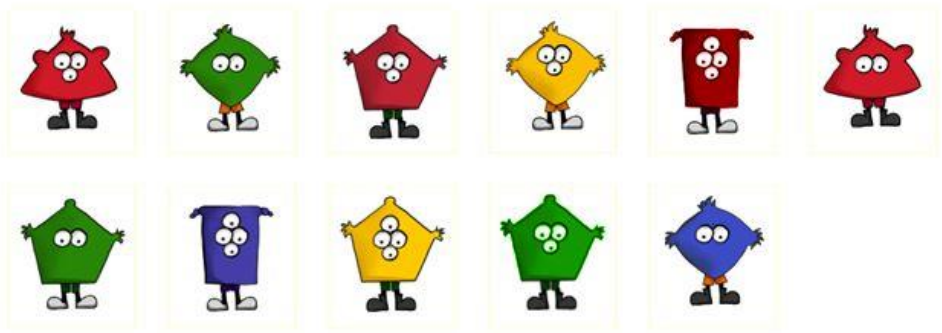


Obr. 9: Séria úloh z prostredia EasyLogo

1.2.5. Vyhodnocovanie podmienok

Ďalším typom príkladov, zameraným na pochopenie porozumeniu významu podmienok, je ich vyhodnocovanie. Úlohy tohto typu nemusia byť priamo spojené s programovaním, skôr majú naučiť dieťa pochopiť, čo je podmienka a na základe nej vyhodnotiť, čo ju spĺňa. V programovacom prevedení môžeme tento typ úlohy nájsť ako súčasť predchádzajúcich – napr. pozbieraj iba červené kvety alebo pohybuj sa iba po zelených políčkach.

Ukážkový príklad na Obr. 10 je zo súťaže iBobor (Bobrík 2013/2014), pozri [8]. Žiaci mali označiť potvorky, ktoré majú tmavé topánky a tri oči.

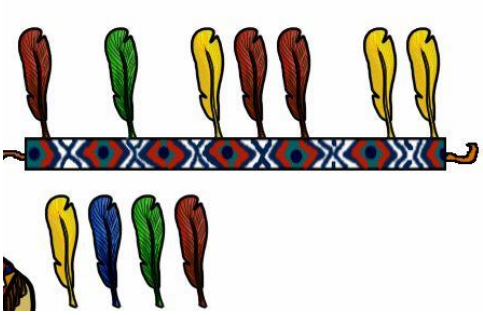


Obr. 10: Úloha zo súťaže iBobor

1.2.6. Hľadanie vzoriek

Tento typ príkladov je dôležitý pre pochopenie cyklov – žiaci majú nájsť opakujúcu sa vzorku, ktorá pri programovaní predstavuje postupnosť krokov. Tento príklad sa dá najprv použiť v podobe pripraveného kódu, v ktorom žiaci menia len počet cyklov, alebo vnútro cyklu, aby pochopili, ako funguje a ako každý pridaný krok alebo počet opakovaní zmení celý program. Neskôr môžu postupne vytvárať vlastné cykly – najprv vnútro cyklu, potom počet opakovaní.

Príklad je zo súťaže iBobor (Bobrík 2011/2012), pozri [8]. Úlohou bolo doplniť pierka do čelenky, pričom zo zadania vyplývalo, že sa v čelenke pravidelne opakovali farby v nejakom poradí.



Obr. 11: Úloha zo súťaže iBobor

1.3. Zásady vyučovania programovania

Podľa [9] existuje 9 základných zásad, ktoré by sa mali využívať pri vyučovaní programovania. Tieto zásady je dôležité poznať a programovacie prostredie by malo byť čo najviac prispôsobené, aby učiteľ tieto zásady mohol používať.

- *Čo najskôr umožniť tvorbu programov* – v mini jazyku je táto zásada splnená hneď, ako sú vysvetlené základné možnosti jazyka a prostredia. Typickým príkladom je okruh úloh z 1.2.6.
- *Nepredbiehať – nepoužívať prvky jazyka, ktoré ešte neboli vysvetlené* – dôležité je, aby úlohy po sebe nasledovali v logickom slede podľa zložitosti konštrukcie.
- *Informácie je potrebné podávať po malých častiach* – pri vyučovaní žiakov na prvom stupni základnej školy je práve táto zásada dôležitá na pochopenie programovania. Každý prvok z programovacieho jazyka by teda mal obsahovať viacero podúloh – od najjednoduchšej po zložitejšie, ktoré postupne vysvetlia preberanú tému.
- *Žiak musí programy nielen vytvárať, ale aj ladiť* – podmienka ladenia programu sa dá pri mini jazykoch splniť dopredu pripraveným programom, do ktorého budú musieť žiaci niečo doplniť, prípadne z neho niečo vymazať alebo niečo zmeniť. A zároveň treba žiakom umožniť upravovať svoje programy, spúšťať ich postupne, po krokoch a tak hľadať vlastné chyby.

- *Sprievodné príklady musia vyžadovať aktívne použitie nových poznatkov* – pri príkladoch precvičujúcich nejakú tému by mali mať žiaci nielen možnosť upravovať existujúci program, ale aj vytvoriť vlastný.
- *Príklady musia byť zaujímavé* – pre žiakov v tomto veku sú zaujímavé úlohy obsahujúce príbeh, alebo pripomínajúce nejakú počítačovú hru.
- *Riešenia nesmú byť príliš zašumené* – t. j. nepoužívať v programe konštrukcie, ktoré deti nepoznajú, alebo ani nepotrebujú poznať.
- *Od začiatku treba žiakom vštepovať zásady moderného programovania.*
- *Predkladať žiakom riešenia netriviálnych problémov* – po zvládnutí základných poznatkov, by mali mať žiaci možnosť vyskúšať vedomosti aj na zložitejších príkladoch.

1.4. Zásady pri tvorbe softvéru pre deti

V článku Sonie Chiasson a Carla Gutwina „Design Principles for Children´s Technology“, pozri [10], autori zadefinovali niekoľko princípov a zásad, ktoré by sa mali využívať pri tvorbe softvéru pre deti. Zo všetkých vyberáme tie najdôležitejšie pre túto prácu.

Prostredie programu by malo byť **grafické** – vizuálne prostredie s minimom textu sú pri softvéri pre deti najlepšou voľbou. Z toho vyplýva aj používanie čo najľahších inštrukcií, v ktorých sa treba vyhýbať dlhým textom, pretože ten deti v tomto veku nemusia pochopiť. Inštrukcie by tak mali byť ľahké na pochopenie a hlavne aj zapamätanie.

V rámci programu je dôležité používať **špecifické ikony**, ktoré deti poznajú (napr. červená ikona stop zastaví program). Navigácia a ovládanie sa tak stane intuitívnym a deti sa ho nemusia dlho učiť.

Pre detských užívateľov je, na rozdiel od dospelých ľudí, potrebná **okamžitá reakcia** na ich akcie a dôležitá je aj **spätná väzba** – upozornenie, že počítač čaká na vstup alebo ho práve vyhodnocuje. Prospešné je aj zvýrazňovanie práve prebiehajúcich častí programu.

Užitočné je **využívanie animácií** – či už vo forme animovanej figúrky, ktorá s dieťaťom komunikuje, alebo animácie na obrazovke. Deti v tomto veku si nevedia predstaviť abstraktné veci, preto je dôležité používanie konkrétnych pojmov a vecí.

Program by mal prechádzať viacerými fázami – od najjednoduchších úkonov až po najzložitejšie, aby sa dieťa postupne naučilo, ako s ním pracovať.

Dlhé rozšírené menu nie sú pre deti vhodné, malo by byť preto jednoduché a čo najkratšie. Vhodná je aj možnosť vrátiť sa v programe späť, aby dieťa bolo schopné zistiť, kde bolo predtým a malo možnosť návratu.

2. Špecifikácia systému

V tejto kapitole sa budeme venovať špecifikácii základných požiadaviek na výsledný systém, možnostiam jeho použitia a dôležitým prvkom, ktoré musí obsahovať.

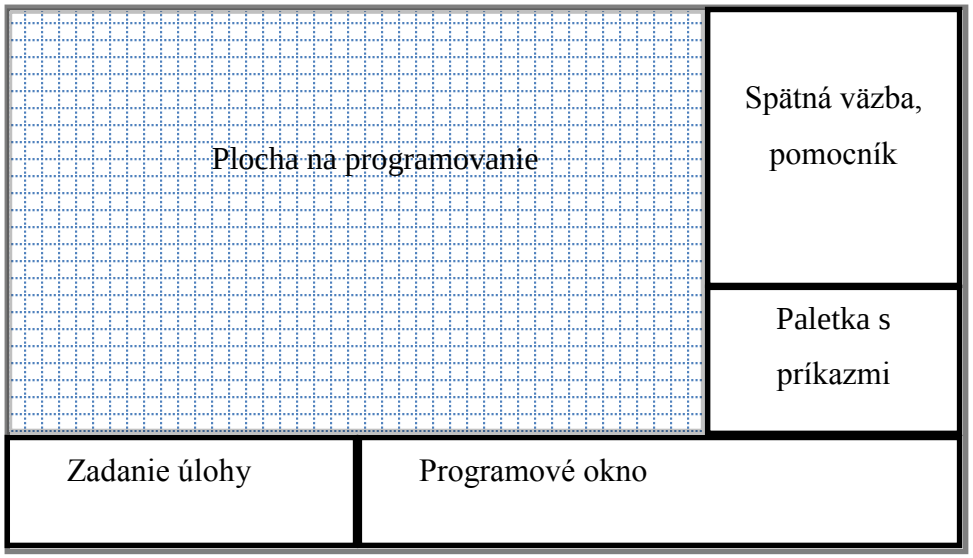
2.1. Edukačná časť

Edukačná časť softvéru je určená žiakom prvého stupňa základnej školy, preto by mala byť vytvorená podľa zásad spomenutých v 1.3. a zároveň spĺňať zásady na vyučovanie programovania. Ako inšpirácia pre túto časť systému budú slúžiť programovacie jazyky spomenuté v prvej kapitole.

2.1.1. Základné prvky edukačnej časti

Základnými prvkami edukačnej časti sú:

- plocha, v ktorej sa budú vykonávať príkazy – najlepšie štvorčeková sieť
- programové okno – časť obrazovky, do ktorej budú žiaci zadávať príkazy a vytvárať program
- paleta s príkazmi – ikonky príkazov, ktoré sa dajú zadať do programového okna
- dialógová časť – obsahujúca zadanie úlohy a komunikáciu s dieťaťom, prípadne pomocníka a prepínanie medzi úlohami
- informácie – textová alebo obrazová informácia o tom, kde v programe sa dieťa nachádza, spätná väzba z riešenia úloh (čas, body alebo nejaká odmena)



Obr. 12: Návrh edukačnej časti

2.1.2. Funkcie edukačnej časti

Edukačná časť softvéru má spĺňať tri základné funkcie – prvou je umožniť žiakovi vytvárať a upravovať programy, druhou je zobrazovať jeho riešenie a treťou funkciou je vyhodnocovanie riešení.

Keďže je softvér určený pre žiakov, ktorí sa len zoznamujú s programovaním, bude v ňom viacero druhov ovládania – tzv. priamy režim (ovládanie šípkami na klávesnici), režim pohybu po prekážku (postavička sa pohybuje daným smerom, kým nepríde k prekážke, alebo sa nezmení jej smer iným príkazom) a potom programátorský mód, v ktorom bude žiak priamo vytvárať programy a upravovať ich. Pohyb v programátorskom móde sme kvôli rôznorodosti úloh rozdelili do dvoch režimov – absolútny pohyb (príkazy: krok dopredu, krok dozadu, krok vpravo, krok vľavo) a relatívny pohyb (príkazy: krok dopredu, otoč sa doprava, otoč sa doľava).

Vytváranie a upravovanie programov bude riešené v programovom okne a príkazy budú z palety s príkazmi. Dôležité je, aby žiak nemohol vytvárať syntakticky zlé programy, a zároveň, aby v palete neboli zobrazené príkazy, ktoré žiak ešte neovláda.

Zobrazovanie riešenia môže byť postupné – v ploche sa zobrazuje každý krok, hneď, ako ho žiak zadá do programovacieho okna. Zároveň však pri niektorých príkladoch môže učiteľ vyžadovať, aby žiak najprv vytvoril postupnosť krokov a tá sa vykonala až po spustení špeciálnym tlačidlom.

V prípade, že žiak potrebuje poradiť, sa v programe nachádza aj pomocník, ktorý mu navrhne ďalší krok. Aplikácia bude mať v sebe zabudovaný aj export výsledkov žiaka (či a kde robil chyby, koľko príkladov vyriešil, koľko mu to trvalo...).

Vyhodnocovanie riešení bližšie popisujeme v časti 2.4. a export výsledkov v časti 2.5.

Obr. 13: Diagram možností použitia žiackeho módu

2.2. Učiteľská časť

Učiteľská časť je určená učiteľom na vytváranie a generovanie úloh pre žiakov. Mala by byť priamou súčasťou aplikácie a ponúkať prehľadné možnosti používania.

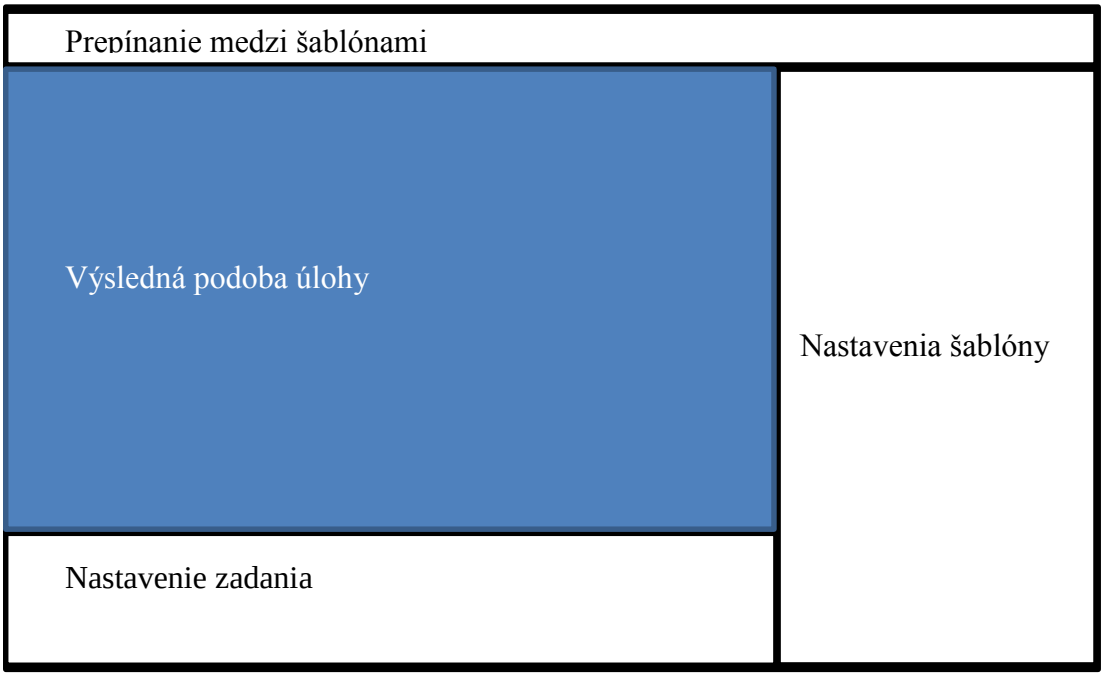
Kvôli prehľadnosti bude rozdelená na karty podobné tým v internetových prehliadačoch.

2.2.1. Základné prvky učiteľskej časti

Základnými prvkami učiteľskej časti softvéru sú:

- Samostatná karta s výberom šablón a ich popisom.
- Panel s možnosťami pre danú šablónu – dôležitý je dostatočný popis jednotlivých možnosti a ľahké zmeny ich nastavení.
- Panel so spoločnými nastaveniami pre všetky šablóny – nastavenie zadania, obrázku postavičky...

- Menu – na rozdiel od edukačnej časti určenej pre žiakov, menu v učiteľskej časti môže byť dlhšie a ponúkať väčšie množstvo možností, stále by však malo byť prehľadné.
- Panel, kde sa zobrazí vygenerovaná úloha, s možnosťou dodatočného editovania



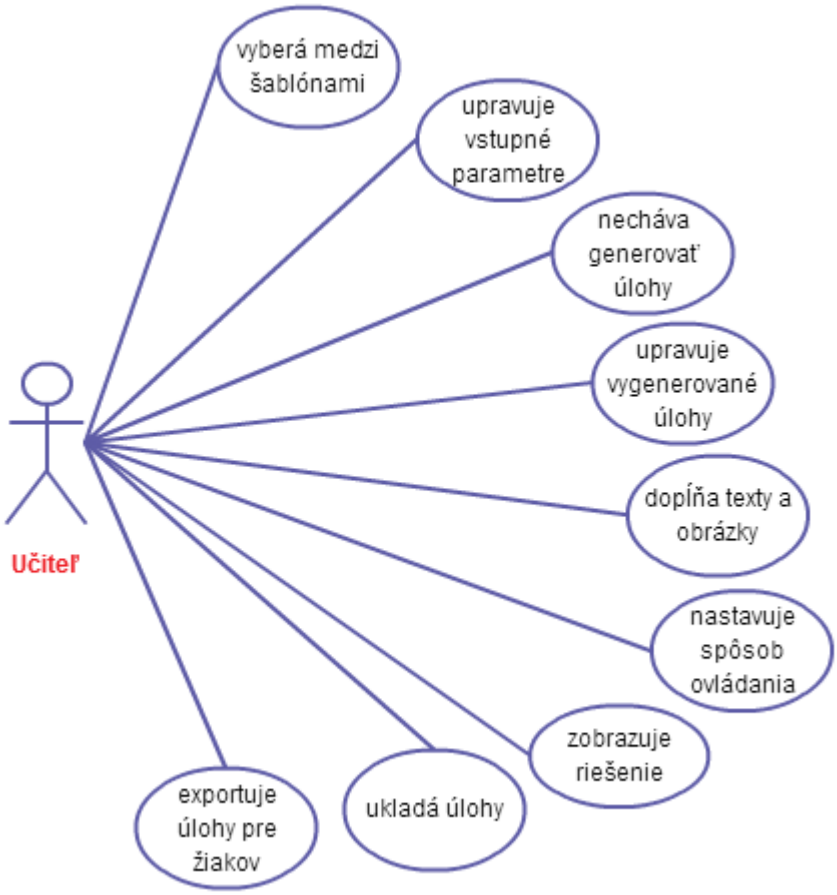
Obr. 14: Návrh učiteľskej časti

2.2.2. Funkcie učiteľskej časti

Učiteľská časť bude obsahovať najdôležitejšiu časť funkcionality tohto softvéru. V tejto časti musí mať učiteľ možnosť na základe preddefinovaných kritérií nechať vygenerovať úlohy z jednotlivých šablón. Pri vytváraní úloh musí mať možnosť ju prispôsobiť pre svojich žiakov (napr. vymyslieť vlastný príbeh, pridať do prostredia vlastnú postavičku atď.). Každú takto vytvorenú úlohu vie softvér upraviť a vygenerovať rôzne jej podoby (napr. iná počiatočná poloha postavičky, iná cesta do cieľa atď.). Vygenerované úlohy by mali mať aspoň jedno riešenie a mali by sa dať ľahko exportovať pre žiakov a takisto ukladať na neskoršie použitie pre učiteľa. Učiteľ vie svoje uložené šablóny v programe otvárať, upravovať a meniť, a tak z jednej úlohy môže po viacerých úpravách vyexportovať niekoľko úloh pre žiakov (napr. zmena ovládania, alebo vyhodnocovania úlohy).

Pri exportovaní je potrebné, aby mal učiteľ možnosť vytvoriť aj sadu úloh, ktoré budú žiaci riešiť za sebou. Učiteľ teda vyberie vytvorené úlohy, ktoré sa majú do sady pridať, a určí poradie ich riešenia. Vie učiť, či žiaci môžu úlohy prepínať (a teda ich riešiť

v ľubovoľnom poradí), alebo je poradie prísne stanovené a systém pustí žiaka na ďalšiu úlohu až keď bude mať hotovú predchádzajúcu.



Obr. 15: Diagram možností použitia učiteľského módu

2.3. Návrh jednotlivých šablón

V tejto časti sa budeme venovať bližšej špecifikácii jednotlivých šablón a možnostiam ich editácie. Zadefinujeme množinu vstupných údajov a spoločných znakov.

Cieľom nášho softvéru je umožniť učiteľovi vytvárať ľahko a rýchlo rôzne úlohy, ktoré majú podobné znaky, ale nie sú úplne totožné. Keďže cieľovou skupinou sú najmladší žiaci, je dôležité, aby boli úlohy vizuálne príťažlivé a obsahovali viacero herných prvkov. Základom každej šablóny bude z tohto dôvodu možnosť nahrania vlastných obrázkov, prípadne zvukov a vymyslenie vlastného príbehu.

Program bude ponúkať galériu obrázkov a zvukov, ktorú si bude môcť učiteľ ľubovoľne rozšíriť a upraviť, aby mohol pripraviť zaujímavé úlohy.

Ďalším dôležitým prvkom by malo byť prednastavenie režimu ovládania – ak učiteľ chce žiakov najprv zoznámiť s prostredím a ukázať im, ako funguje, môže zvoliť najjednoduchší mód ovládania šípkami, ktorý žiaci poznajú z mnohých počítačových hier. Sťažením môže byť režim ovládania pohybu po prekážku. V prípade, že učiteľ bude chcieť, aby žiaci programovali, zvolí programátorský mód a zároveň vyberie typ pohybu.

Zadania úloh zostanú rovnaké, mení sa iba spôsob ich ovládania a vyhodnocovania, takže rovnakú úlohu vieme žiakom zadať vo všetkých režimoch ovládania a tým im ukázať rozdiely medzi nimi.

2.3.1. Pohyb postavičky do cieľa

Typ tejto úlohy je bližšie špecifikovaný v 1.2.1. Základnými údajmi dôležitými pre generovanie úloh tohto typu je veľkosť poľa, v ktorom sa postavička pohybuje.

Učiteľ si vie navoliť jednotlivé rozmery, stanoviť začiatok a koniec cesty, alebo jej dĺžku. Na základe týchto vstupných parametrov vie systém vygenerovať viacero úloh – zmení sa počiatočná a/alebo konečná pozícia, s tým, že náročnosť úlohy by mala zostať približne rovnaká (podobná dĺžka trasy, podobná náročnosť ovládania).

Systém vygeneruje možnú cestu, ale zároveň ponúkne možnosť, aby učiteľ predvolil cestu (napr. zobrazenú inou farbou políčok). Predvolená cesta znamená iba jedno riešenie úlohy, bez nej bude pre každú úlohu existovať viac riešení.

Dôležitým parametrom je možnosť ovládania – to sa nastaví žiakovi hneď pri zapnutí úlohy.

Na základe takto vytvoreného modelu úlohy – dĺžka cesty, náročnosť, ovládanie, atď. – vie systém generovať jej podobné úlohy. Hlavná postavička a príbeh zostáva, mení sa rozloženie plochy a tým pádom aj postupnosť príkazov, ktoré musí žiak použiť na vyriešenie úlohy.

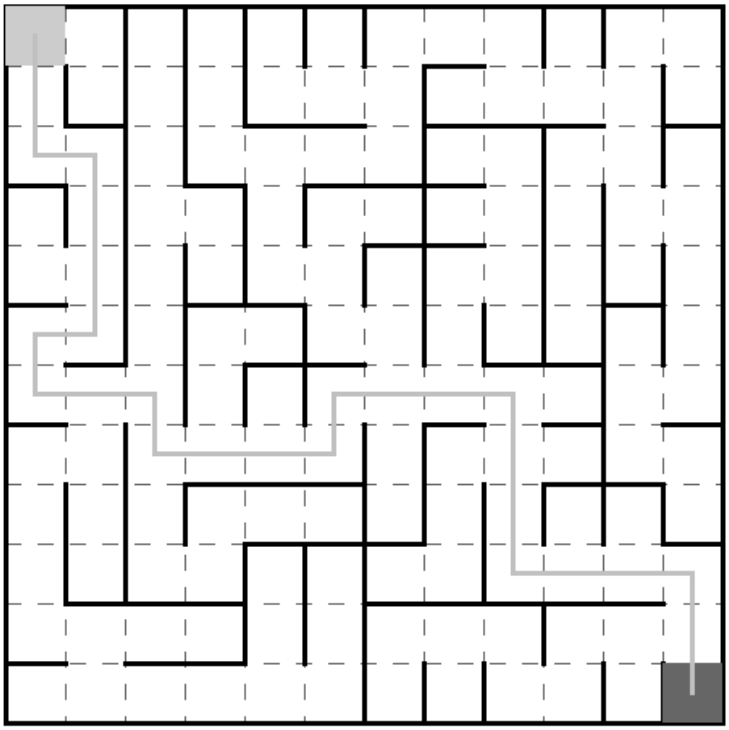
2.3.2. Cesta v bludisku

Cesta v bludisku, typ tejto úlohy je bližšie špecifikovaný v 1.2.2, je podobná šablóne z 2.3.1., rozdielom je väčšia množina nastavení.

Základom sú stále rozmery plochy, v ktorej sa bude bludisko nachádzať. Softvér vygeneruje labyrint, ktorý učiteľ môže ešte ďalej upravovať – meniť počiatočnú alebo

konečnú pozíciu, alebo pridávať a odoberať hrany v bludisku. Zároveň by mal mať stále prehľad, či existuje riešenie.

Učiteľ vie vybrať aj to, či má žiak hľadať najkratšiu cestu, alebo akúkoľvek.



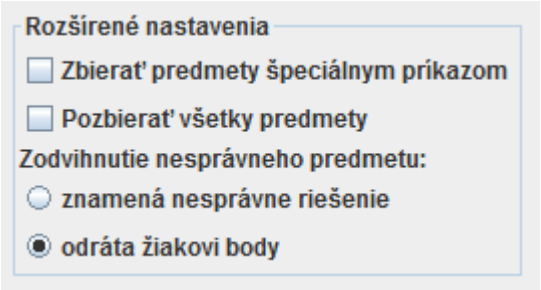
Obr. 16: Návrh vygenerovaného labyrintu

2.3.3. Zbieranie predmetov

Keďže pri úlohách zameraných na zber premetov, pozri1.2.3., sa postavička pohybuje po ploche alebo bludisku, nastavenia zostávajú podobné ako v predchádzajúcich dvoch šablónach, pribúdajú iba nastavenia obrázkov pre jednotlivé predmety a ich počet.

Zároveň je dôležité, aby učiteľ vedel nastaviť, či zbieranie predmetov bude realizované iba prechodom cez políčko predmetu, alebo nejakým špeciálnym príkazom (čo bude žiak musieť použiť, aby sa mu podarilo pozbierať predmety). Pre zložitejšie typy úloh sa budú dať nastaviť dva druhy predmetov – dobré a zlé – jedny, ktoré má žiak zbierať, a druhé, ktorým sa vyhnúť.

Pre vyhodnocovanie bude dôležité, aby učiteľ nastavil, či má žiak vyzbierať všetky predmety, alebo len niektoré a pri dvoch druhoch, či prechod po zlom predmete znamená zle vyriešenú úlohu, alebo iba odráta počet pozbieraných predmetov.



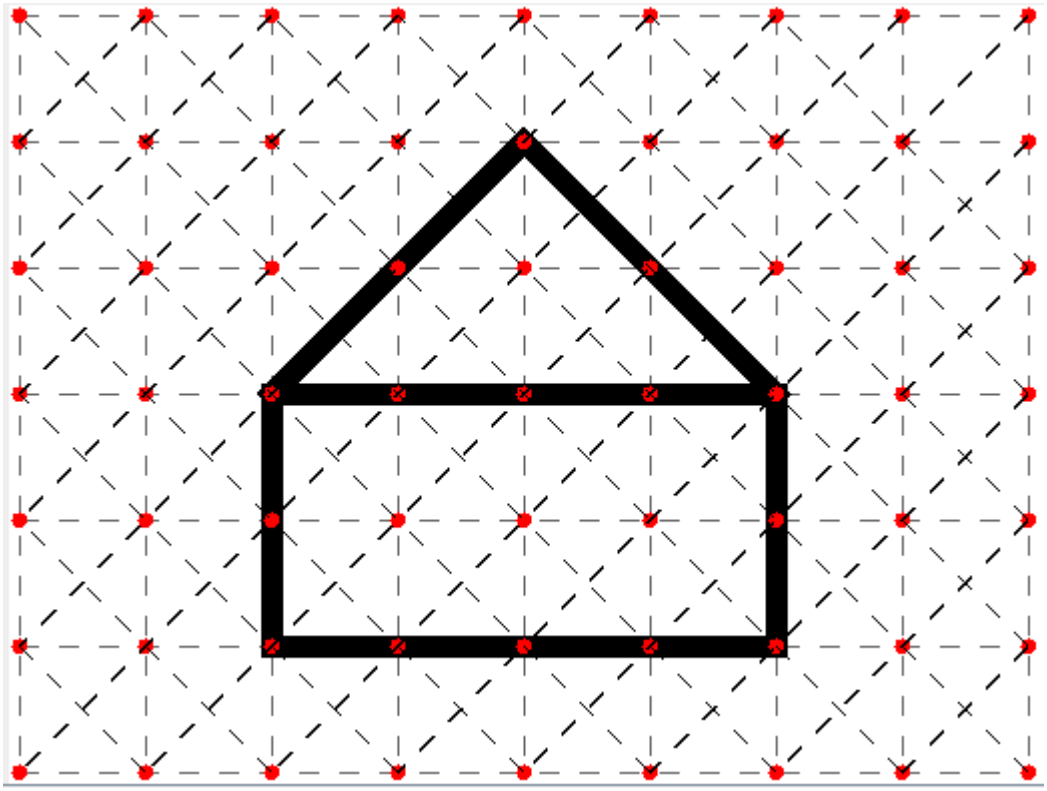
Obr. 17: Návrh rozšírených nastavení

2.3.4. Kreslenie tvarov

Táto šablóna vychádza zo špecifikácie úloh z 1.2.4., preto je potrebné, aby sa v jej rámci dala vytvoriť skupina príkladov. Keďže žiak sa musí naučiť rozdeliť úlohu na menšie časti, malo by byť zadanie prispôsobené práve na postupnosť menších čiastkových úloh, ktoré sa následne dajú dokopy.

Priamo v programe bude výber základných geometrických tvarov, z ktorých bude vedieť učiteľ vybrať postupnosť, v akej sa budú kresliť a následne ich výsledné spojenie. Samozrejmosťou je možnosť návrhu vlastného tvaru a to vo dvoch variantoch – učiteľ vytvorí program, ktorý nakreslí výsledný tvar a ten sa bude porovnávať so žiackym riešením, alebo priamo do plochy nakreslí obrázok, ktorý má žiak zostrojiť. Zároveň vie nastaviť, či je dôležité, aby žiak používal cykly, alebo stačí, ak použije iba jednoduchú postupnosť krokov.

Pri tejto šablóne si učiteľ vie vybrať, ako sa má zobrazovať predkreslená trasa – či iba ako čiara, alebo má byť v čiarach naznačený aj smer.



Obr. 18: Návrh plochy pre učiteľa, určenej na priame kreslenie

2.3.5. Vyhodnocovanie podmienok

Šablóna, pozri 1.2.5., skladá v sebe predchádzajúce šablóny. V rámci tejto šablóny si učiteľ vie vybrať, ktorú zo šablón 2.3.2. alebo 2.3.3. použije ako predlohu pre úlohu.

V prípade šablóny 2.3.2., pohyb po bludisku, môže nastaviť, aké podmienky má žiak používať (napr. Ak si pri stene, otoč sa. Ak je pred tebou jama, choď naspäť...) Systém vygeneruje úlohu podľa šablóny 2.3.2. a do programovacieho módu pridá predpripravené podmienky, ktoré bude musieť žiak použiť.

Ak použije ako predlohu šablónu z 2.3.3., bude v nej prednastavené dvíhanie predmetov špeciálnym príkazom. Dôležité je preto mať aspoň dva druhy predmetov a vybrať, ktoré má žiak zbierať a podľa toho spraviť podmienky – Ak si na políčku so zeleným kvietkom, zodvihni ho. Ak si na políčku s červeným kvietkom, nechaj ho na zemi.

2.3.6. Hľadanie vzoriek v programoch

V tejto šablóne sa bude používať predpripravený kód, ktorý bude môcť žiak modifikovať. Učiteľ zadá kód, ktorý budú jeho žiaci vidieť a môže zadať aj výsledný obrázok, aby vedeli, ako majú program upraviť.

Zadá, či treba upravovať počet opakovaní alebo telo cyklu, prípadne oboje. Zároveň môže vytvoriť program, v ktorom sa opakujú určité časti kódu. Tie má žiak nájsť a na základe nich vytvoriť cykly.

2.4. Vyhodnocovanie jednotlivých úloh

Pri každej z úloh vie učiteľ nastaviť, podľa akých parametrov má systém úlohy vyhodnocovať. Samozrejmosťou sú základné logické pravidlá – postavička nemôže vyjsť mimo plochy, prechádzať cez steny a brať predmety z prázdneho políčka. Pri programovacích príkazoch je ošetrený počet opakovaní cyklu minimálnym počtom dvakrát. Systém na základe zadefinovaných parametrov vyhodnocuje čiastkové riešenia a upozorňuje žiaka na chyby – hlavne logické (napr. Nemôžem prechádzať cez steny.) a zároveň ho upozorní, ak sa dostane do stavu, v ktorom už úlohu nevyrieši.

Pri vyhodnocovaní celého riešenia zisťuje, či žiak splnil všetky povinné podúlohy, ktoré mu zadal učiteľ a či príklad riešil tak, ako bolo zadané (napr. či pri kreslení používal cykly, či našiel najkratšiu cestu v bludisku atď.). Chyby žiakovi oznámi, prípadne vysvieti chybné časti programu. Softvér vie ukázať žiakovi vzorové správne riešenie, to vie vygenerovať sám, alebo ho priamo k úlohe vie vložiť aj učiteľ.

2.5. Export výsledkov

Dôležitou funkcionalitou, poskytujúcou spätnú väzbu učiteľovi, je export žiakových výsledkov do tabuľky. V tabuľke by mal učiteľ vidieť, ktorú sadu úloh žiak riešil, ako obstál v jednotlivých úlohách, kde robil chyby, koľkokrát použil pomocníka a ako dlho mu trvalo riešenie.

Tabuľka má slúžiť učiteľovi na vyhodnotenie hodiny a zistenie nedostatkov. Preto je potrebné, aby ku každej úlohe bol v tabuľke uvedený aj spôsob ovládania a v prípade nesprávneho riešenia aj priamo program, ktorý vytvoril. Učiteľ tak uvidí, ktorá oblasť je pre žiaka problematická a zároveň vie porovnávať výsledky všetkých žiakov.

2.6. Návrh mini jazyka

Keďže má byť softvér určený pre žiakov na prvom stupni základnej školy, mini jazyk bude kartičkový – väčšinu príkazov bude znázorňovať ikonka.

Pri absolútnom pohybe by ikony jednotlivých kartičiek mali čo najviac pripomínať šípky na klávesnici. Pri relatívnom pohybe sa na kartičkách budú nachádzať natočené šípky, aké poznáme z väčšiny editorov. Pri týchto šípkach sa bude nachádzať aj číselná informácia o otočení (45° alebo iba šípka pre 90° otočenie).

Jednotlivé sady kartičiek by mali byť farebne odlíšené, aby žiaci na prvý pohľad vedeli určiť, ktorý typ pohybu práve programujú. Výnimku tvorí kartička s obrázkom ruky – slúžiaca na zbieranie predmetov, a kartička na tvorbu cyklov.

Kartička znázorňujúca cyklus by mala obsahovať textovú informáciu o počte opakovaní, ktoré by sa mali dať meniť – pomocou ikoniek plus a mínus priamo na kartičke. Kartička znázorňujúca cyklus by mala byť vyššia, ako zvyšné kartičky a mala by ohraničovať určitú plochu, do ktorej bude možné vkladať jednotlivé príkazy – vnútro cyklu.

Kartičky s podmienkami by sa mali skladať z dvoch častí – podmienky „Ak...“ a časti, do ktorej žiak priamo vloží jednotlivé príkazy. Podmienková časť by mala byť priamo zadaná a nemenná. Príkazovú časť môže žiak upravovať – pridávaním, alebo odoberaním kartičiek.

3. Implementácia

V časti implementácia popisujeme nami implementované časti softvéru, použité algoritmy a výslednú podobu aplikácie.

3.1. Všeobecný prehľad systému

Softvér sme nazvali GEVIN (Generátor úloh na vyučovanie informatiky) a je vytvorený v programovacom jazyku Java [12] v prostredí Eclipse, pozri [13], ktoré sme vybrali kvôli jeho multiplatformovému využitiu. Softvér sme testovali na operačných systémoch MS Windows – Vista, 7, 8 a Ubuntu.

Aplikácia má podobu appletu (prípona .jar) a to umožňuje jej spustenie jednak lokálne na počítači užívateľa, ale zároveň aj na internete, čo môže vyriešiť problémy s inštaláciou softvéru.

Systém je rozdelený na dve časti – učiteľskú a žiacku. Po zapnutí aplikácie sa spustí úvodná karta – do učiteľskej sa učiteľ dostane po kliknutí na tlačidlo „Som učiteľ“ a zadaní hesla, na vstup do žiackej časti stačí iba vybrať tlačidlo „Som žiak“.

Vzhľad softvéru je pre obe časti približne rovnaký. Žiacky mód je učiteľovi priamo dostupný z jeho módu, aby učiteľ mohol otestovať ním vytvorené úlohy. K programu existuje aj knižnica s obrázkami, ktoré sú prednastavené pri šablónach a môžu tak uľahčiť učiteľovi vytváranie úloh.

Obr. 19: Úvodná karta

3.2. Učiteľský mód

V rámci učiteľského módu sme vytvorili niekoľko nezávislých kariet s jednotlivými šablónami. Úvodná karta, ktorá sa prihlásenému užívateľovi zobrazí, obsahuje popis jednotlivých šablón a uložené úlohy.

Každá zo šablón ponúka možnosť uloženia, ktoré slúži zároveň aj na export. V prípade, že učiteľ chce vytvoriť sadu úloh, na úvodnej karte jednotlivé vygenerované úlohy zoradí a uloží do súboru.

3.2.1. Cesta v bludisku

Po kliknutí na kartu „Labyrint“ sa užívateľovi zobrazí plocha s nastaveniami pre úlohu Cesta v bludisku, pozri 2.3.2.

Po nastavení rozmerov plochy systém vygeneruje labyrint – štvorčekovú plochu so stenami. Na vygenerovanie labyrintu sme použili mierne modifikovaný Primov algoritmus, pozri [14]. Primov algoritmus je greedy algoritmus slúžiaci na nájdenie kostry neorientovaného grafu. Pracuje na jednoduchom princípe – vyberie si počiatočný vrchol a postupne vytvára strom spájaním vrcholov s minimálnymi hranami. Algoritmus sme pre potreby generátora upravili – hrany nie sú ohodnotené a nájdené hrany vymazávame.

Na začiatku máme vytvorený graf, kde každé políčko reprezentuje vrchol a hrany grafu reprezentujú spojenia dvoch susedných políčok. Ak sa medzi dvoma vrcholmi nachádza hrana, znamená to, že vieme prejsť z jedného do druhého. Ak tam hrana nie je, medzi políčkami je stena.

V počiatočnej fáze algoritmu sa náhodne vyberie jedno políčko a hrany, ktoré z neho vychádzajú, sa pridajú do zásobníka. Kým nie je zásobník prázdny, vyberáme z neho jednu hranu (i, j). Ak sme už boli vo vrchole j, hranu vymažeme zo zásobníka, inak vytvoríme stenu medzi i a j, t. j. vymažeme hranu (i, j), pridáme do zásobníka hrany z vrcholu j a označíme ho ako navštívený.

```
while (!steny.isEmpty()){
                    Iterator<Stena> it = steny.iterator();
                    Stena wall = it.next();
                    Integer I = wall.i;
                    Integer J = wall.j;
                    if (!visited.contains(J)){
                          plocha.get(J).remove(I);
                          plocha.get(I).remove(J);
                          for (int i=0; i<plocha.get(J).size(); i++){
                                Stena s = new Stena(J, plocha.get(J).get(i));
                                steny.add(s);
                          }
                          visited.add(J);
                    }
                    else{
                          steny.remove(wall);
                    }
  }
```

V takto vytvorenom labyrinte existuje cesta medzi ľubovoľnými dvoma políčkami. Program prednastaví ako začiatočnú pozíciu postavičky políčko [0,0] – začiatočné políčko vľavo hore, a ako konečnú pozíciu políčko vpravo dole – [šírka plochy-1, výška plochy-1].

Používateľ môže labyrint ešte upravovať – kliknutím na stenu sa táto stena vymaže, kliknutím na prerušovanú čiaru medzi políčkami sa medzi políčka stena pridá. Používateľ vie takisto zmeniť pozíciu počiatočného a koncového políčka.

Program automaticky hľadá najkratšiu cestu medzi začiatočným a koncovým políčkom a zobrazuje ju graficky ako sivú čiaru spájajúcu tieto políčka. Ak sa čiara nezobrazuje, znamená to, že medzi políčkami neexistuje cesta.

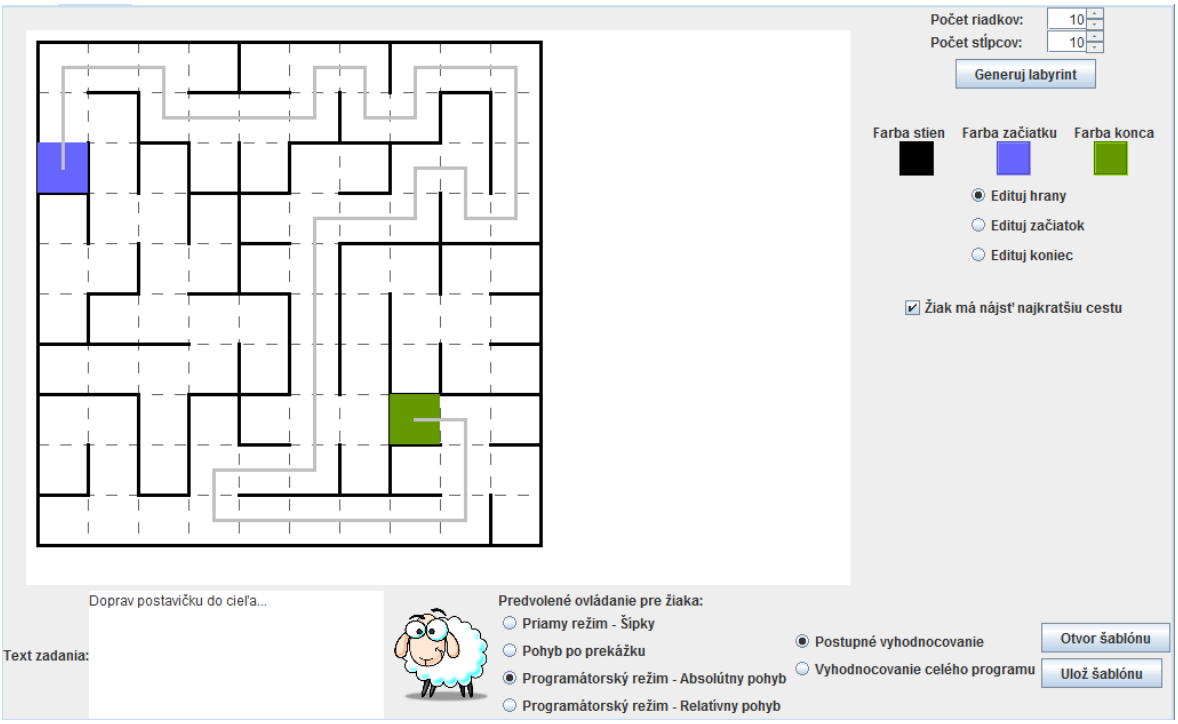
Pri hľadaní cesty medzi začiatočným a koncovým políčkom zisťujeme pre každé políčko vzdialenosť od počiatočného. Na ohodnotenie vzdialeností používame funkciu *navstivSusedov*.

```
privatevoid navstivSusedov(Integer policko, int vzdial, int koniec){
      if ((susedia.get(policko)==null) || (susedia.get(policko) > vzdial)){
            ArrayList<Integer> susedne = cesty.get(policko);
            if (susedne.isEmpty()){
                  return;
            }
            navstiveniSusedia.add(policko);
            susedia.put(policko, vzdial);
            for (int i=0; i<susedne.size(); i++){
                  Integer s = susedne.get(i);
                  navstivSusedov(s, vzdial+1, koniec);
            }
      }
  }
```

Ak sme vo vstupnom políčku ešte neboli, alebo je má nastavenú vyššiu hodnotu ako je vstupná (touto podmienkou zabezpečujeme, že v prípade, ak sa do aktuálneho políčka vieme dostať aj kratšou cestou, zapísaná bude tá), navštívime ho a rekurzívne spustíme algoritmus aj pre políčka s ním susediace s tým, že hodnota vzdialenosti od začiatočného políčka sa zväčší o 1.

Z takto ohodnoteného grafu hľadáme cestu od koncového políčka späť k začiatočnému a to tak, že spájame postupnosť klesajúcich čísel. Ak existuje viacero rovnako dlhých najkratších ciest, program zobrazí iba jednu, ale pri vyhodnocovaní sa budú za najkratšie považovať všetky s rovnakou dĺžkou.

Učiteľ vie nastaviť, či majú žiaci nájsť najkratšiu cestu, alebo na dĺžke cesty nezáleží. Program bude potom vyhodnocovať ako správne riešenie každé, v ktorom sa žiak dostane na koncové políčko.



Obr. 20: Náhľad učiteľského prostredia s vygenerovaným labyrintom

3.2.2. Zbieranie predmetov v štvorčekovej sieti

Na karte „Predmety“ môže užívateľ vytvoriť úlohu zadefinovanú v 2.3.3. s tým, že aktuálna verzia aplikácie ponúka pridávanie predmetov do štvorčekovej siete.

Používateľ si vie navoliť rozmery štvorčekovej plochy a počet predmetov, ktoré má program do plochy umiestniť. Na umiestňovanie predmetov sme vymysleli vlastný algoritmus.

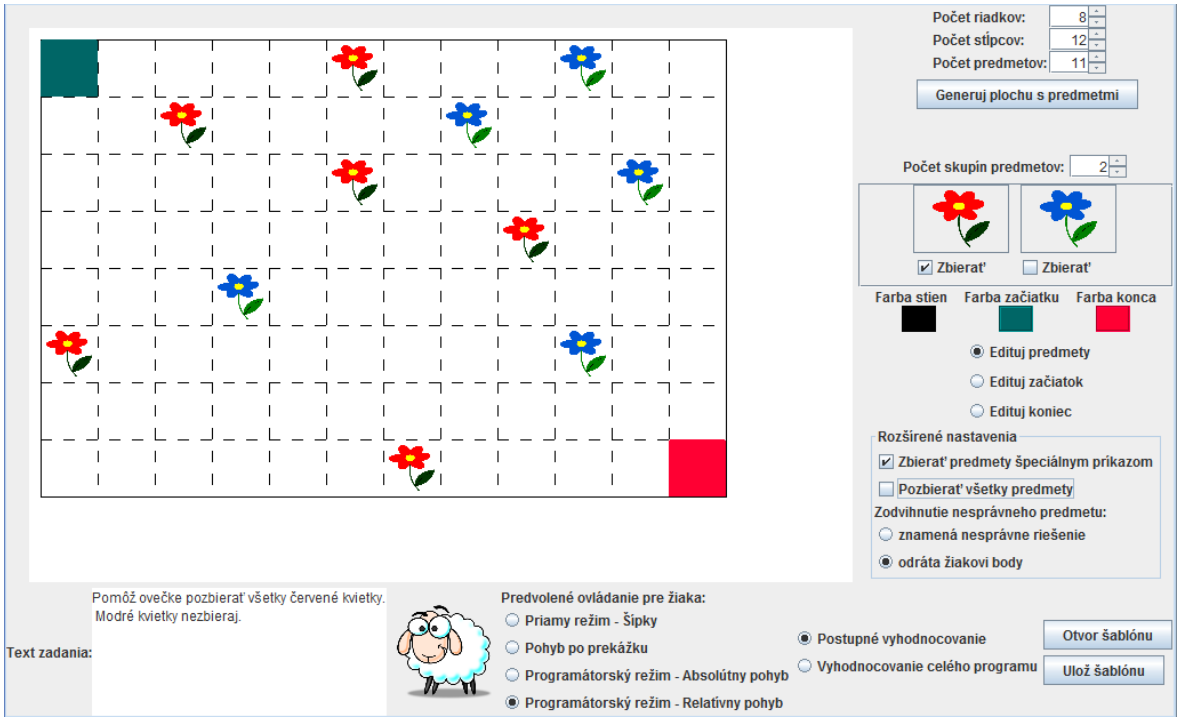
Plocha sa rozdelí na oblasti podľa počtu predmetov – a to aj vo vertikálnom, aj v horizontálnom smere. Veľkosť oblasti je horná celá časť počtu voľných políčok v ploche (x-ový rozmer krát y-ový rozmer mínus dva – začiatočné a koncové políčko) delene počet predmetov. Tým zaručíme dostatočný počet a veľkosť jednotlivých oblastí. Predmety sa náhodne pridávajú do oblastí tak, že pridaním predmetu od oblasti (v oboch smeroch) sa tieto oblasti označia ako obsadené. Zároveň sa ako obsadené označia aj políčka susediace s práve pridaným predmetom.

V prípade, že sú obsadené všetky oblasti, ale ešte nie sú umiestnené všetky predmety, náhodne sa vyberie políčko, ktoré nie je obsadené – t.j. nesusedí so žiadnym predmetom. Takto zabezpečujeme približne rovnomerné usporiadanie predmetov v ploche.

Novo-vygenerované predmety sú zobrazené ako farebné krúžky, používateľ však vie nastaviť počet skupín predmetov (maximálne päť), pre ktoré vie nastaviť obrázok. Predmety v ploche sa automaticky rozdelia do skupín a vykreslia podľa nastaveného obrázku. Tie sú v šablóne preddefinované, no po kliknutí na obrázok si učiteľ vie vybrať vlastný, ktorým ho nahradí.

Pre každú skupinu môžeme nastaviť, či ju majú žiaci zbierať, alebo nie. V tejto šablóne má učiteľ rozšírené nastavenia zbierania predmetov – môže nastaviť, či sa predmety majú zbierať špeciálnym príkazom (v priamom režime pomocou medzerníka, vo zvyšných režimoch kartičkou s obrázkom ruky), či má žiak pozbierať všetky predmety a zvoliť jednu z možností vyhodnocovania pri zodvihnutí nesprávneho predmetu –systém buď žiakovi odráta body, alebo označí riešenie ako nesprávne.

Všetky tieto nastavenia si program ukladá, aby ich vedel pri spustení úlohy v žiackom móde nastaviť. Vyhodnocovanie potom prebieha na základe zadaných kritérií.



Obr. 21: Náhľad učiteľského módu s vygenerovanou plochou s predmetmi

3.2.3. Kreslenie v štvorčekovej sieti

Na vygenerovanie úlohy z 2.3.4. slúži karta „Kreslenie“. Užívateľ nastaví rozmery štvorčekovej plochy, do ktorej sa bude kresliť. Pri kreslení je plocha reprezentovaná ako graf s vrcholmi v rohoch štvorčekov v sieti. Učiteľ môže úlohu vytvárať dvoma spôsobmi – priamym kreslením tvaru, alebo programovaním pomocou kartičiek.

Pri priamom kreslení sa vo vrcholoch štvorčekov zobrazujú červené body. Po kliknutí na vybraný bod sa tento označí v šablóne ako počiatočný a pre používateľa je jeho výber znázornený väčším červeným kruhom. Po označení vrcholu, ktorý je s ním spojený hranou (8 vrcholov, vždy po 45°), sa vytvorí hrana a nový vrchol sa označí ako aktuálna pozícia pera. Program neumožňuje vytvárať hrany medzi nesusednými vrcholmi, umožňuje ale viacnásobný prechod po hrane.

Postupnosť krokov potrebná na nakreslenie daného tvaru sa generuje po každom ťahu a vykresľuje do programového okna. Program ako počiatočné natočenie berie prvý ťah, tak nastaví aj natočenie pera v žiackom režime.

Kroky sú generované z preddefinovaných postupností. Program vyhodnocuje aktuálne natočenie pera a pomocou príkazov „otoč vpravo“ a „otoč vľavo“ (s tým, že každý príkaz má dve možnosti – otočenie o 90° a otočenie o 45°) ho natočí do požadovaného smeru a príkazom „dopredu“ ho posunie o jeden krok v smere natočenia.

Po kliknutí na tlačidlo „zostroj cykly“, program zistí, či sa v postupnosti krokov nachádzajú nejaké opakujúce sa častí a ak áno, tak z nich vytvorí cykly. Pri vytváraní cyklov má väčšiu prednosť počet opakovaní cyklu pred počtom kartičiek v ňom. (Stále však musia byť v cykle aspoň dve kartičky).

Hľadanie opakujúcich častí vykonáva funkcia *vytvorenieCyklu*, ktorá postupne zisťuje opakovanie jednotlivých častí vstupného reťazca. Funkcia *pocetOp* vracia počet opakovaní postupnosti vo vstupnom reťazci. Pre každý podreťazec teda vieme určiť počet opakovaní a funkcia *vytvorenieCyklu* vracia podreťazec, ktorý sa vo vstupe nachádza najčastejšie.

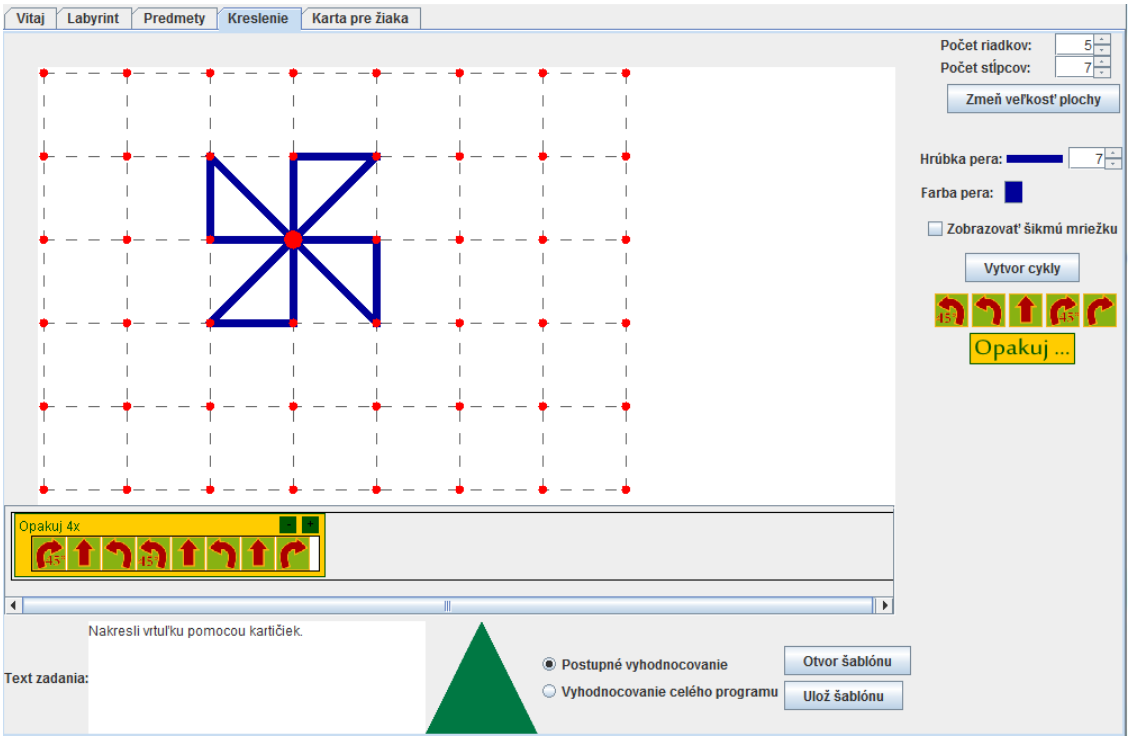
```
public String vytvorenieCyklu(String kod){
		int[] poleOpak = newint[kod.length()/2-1];
		int index = 0;
		for (int i=2; i<=kod.length()/2; i++){
			poleOpak[index] = pocetOp(kod, kod.substring(0, i));
			index++;
		}
		int maxC = 0;
		index = -1;
		for (int i=0; i<poleOpak.length;i++){
			if (poleOpak[i] > maxC){
				maxC = poleOpak[i];
				index =i;
			}
		}
		if (index!=-1){
			String s = kod.substring(0, index+2);
			return s + ";" + maxC;
		}
		return"";
	}
```

Funkciu *vytvorenieCyklu* voláme pre každý reťazec, ktorý vznikne z postupnosti krokov odoberaním znakov zo začiatku reťazca (0 až (dĺžka reťazca/2 – 3) znakov). Takto zistenú najčastejšie opakujúcu sa časť postupnosti nahradíme cyklom.

Pri zadávaní programu má učiteľ rovnaké možnosti ako žiak – používa rovnaké kartičky, cykly a program môže upravovať. Softvér zadaný program vyhodnocuje a obrázok sa vykresľuje po každej zmene.

Samozrejmosťou šablóny je nastavenie farby a hrúbky čiary. Tá bude prednastavená aj v žiackom režime pri kreslení.

Vyhodnocovanie žiackeho riešenia kontroluje zhodu žiakovho obrázku s obrázkom učiteľa. Program zistí, či oba obrázky obsahujú rovnaké body a hrany medzi nimi. Ich postupnosť nemusí byť rovnaká. Tak zabezpečujeme správnosť viac ako jedného riešenia.



Obr. 22: Náhľad šablóny kreslenie s vygenerovanou postupnosťou krokov

3.3. Žiacky režim

Pri vstupe do žiackeho režimu si systém od žiaka vypýta jeho meno. To je potrebné na export výsledkov.

Po kliknutí na tlačidlo „Otvor úlohy“ sa žiak dostane do priečinku so zadaniami. Po vybratí zadania sa toto otvorí v programe. Ak učiteľ vygeneroval sadu úloh, spustí sa prvá z nich.

Pre každú úlohu je prednastavené ovládanie. Žiak vidí, ktoré z ovládaní je práve nastavené a rovnakú informáciu obsahuje aj textové zadanie.

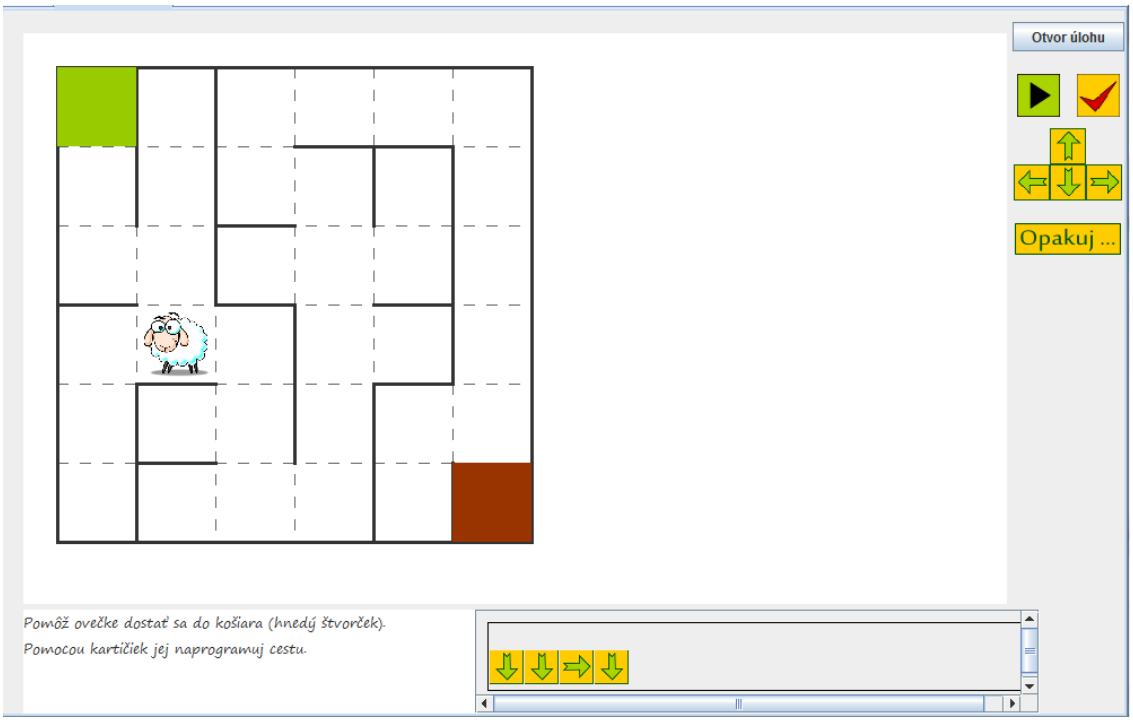
Pri programátorskom režime má žiak k dispozícii kartičky podľa typu pohybu (absolútny alebo relatívny). Po klikaní na ikonky kartičiek sa pridávajú do programového okna, kde ich žiak môže ďalej upravovať – posúvať pomocou drag-and-drop, premiestňovať do alebo

z cyklov, prípadne mazať – potiahnutím kartičky na ikonu koša alebo pravým klikom myši. Softvér neumožňuje žiakovi vytvárať syntakticky zlé programy.

V rámci programátorských módov sme vytvorili interpreter kartičkového jazyka. Ak je nastavené postupné vyhodnocovanie, softvér reaguje na každú zmenu programu. Pri vyhodnocovaní až po vyžiadaní, sa po kliknutí na ikonku „spusti“, spustí krokovač, ktorý postupne ukazuje, ako sa postavička hýbe. Zároveň zvýrazňuje práve prebiehajúci krok.

Program kontroluje správnosť žiackeho riešenia a to priebežne – kontroluje pozíciu postavičky a neumožňuje jej vyjsť mimo plochy, prehádzať cez steny, alebo zbierať predmety z prázdneho políčka. O každom chybnom kroku je žiak informovaný pomocou dialógu.

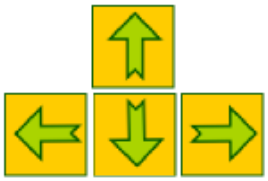
V rámci žiackeho módu je implementovaný aj návrh ďalšieho kroku, o ktorý môže žiak požiadať kliknutím na ikonku otáznika. Tento návrh je riešený hľadaním najkratšej cesty do cieľa z aktuálnej pozície (v prípade labyrintu), vyhľadávaním najbližšieho predmetu a/alebo cieľa (v prípade zbierania predmetov) a hľadaním ešte nevykonaného ťahu pri kreslení. Program žiakovi ponúkne textovú informáciu o možnom ďalšom kroku a zároveň zobrazí ikonku prislúchajúcej kartičky – ak sme v programovacom móde.



Obr. 23: Náhľad žiackeho módu s absolútnym pohybom v labyrinte

3.4. Mini-jazyk

V aktuálnej verzii systému GEVIN sú implementované dva programovacie módy. Absolútny a relatívny pohyb. Pri absolútnom pohybe sú k dispozícii štyri kartičky, ktoré priamo naznačujú smer.



Obr. 24: Kartičky absolútneho pohybu

Kartičky relatívneho pohybu závisia od spustenej úlohy. Pri pohybe v štvorčekovej sieti alebo labyrinte má žiak na výber z troch kartičiek – krok dopredu, otoč sa vpravo, otoč sa vľavo, s tým, že otočenie znamená natočenie o 90°.

Pri kreslení je potrebné, aby žiaci mohli kresliť aj šikmé čiary, preto sa pridávajú ešte dve kartičky – otoč sa vpravo o 45° a otoč sa vľavo o 45°. Pri natočení po diagonále sa krok dopredu ráta ako krok k nasledujúcemu bodu. Dĺžky kroku sa tak líšia, keďže je ale program určený pre žiakov prvého stupňa, toto zjednodušenie je dôležité pre jednoduché ovládanie, pozri aj zdôvodnenia v [5].

Obr. 25: Kartičky relatívneho pohybu

V oboch setoch je pridaná kartička opakuj. Po pridaní do programového riadku sa dá nastaviť počet opakovaní klikaním na štvorčeky plus a mínus priamo na kartičke (prednastavenou a najmenšou hodnou je 2). Pridanie kartičky do cyklu je realizované jednoduchým drag and drop priamo do cyklu.

Pri úlohách na zbieranie predmetov s nastaveným špeciálnym príkazom na zbieranie je pridaná kartička s ikonkou ruky.

Všetky ikony boli vytvorené v grafickom vektorom editore Inkscape, pozri [15].

Obr. 26: Špeciálne kartičky

Systém si ukladá znakovú reprezentáciu jednotlivých kartičiek a takto vzniknutý reťazec delí na jednotlivé kroky a tie potom postupne vyhodnocuje.

3.5. Export výsledkov

Aplikácia dokáže exportovať žiacke výsledky do externého súboru. Žiak na začiatku zadáva do systému svoje meno, preto učiteľ vie určiť, o ktorého žiaka sa jednalo.

Systém exportuje výsledky do formátu CSV, pozri [15]. Zapisuje tam meno žiaka, dátum riešenia a úlohu, alebo sadu úloh, ktoré riešil. Pre každú úlohu sa zobrazí názov úlohy (meno súboru, ktorý učiteľ vytvoril), nastavené ovládanie a kritéria, ktoré do systému zadal učiteľ. V súbore sa nachádzajú aj informácie o žiackom riešení, počet, koľkokrát použil radu od systému, ako dlho mu trvalo riešenie úlohy, výsledná správnosť a v prípade programovacieho ovládania, aj celý program v textovej forme.

Meno žiaka	Tomáško
Úloha	labyrint4.lab
Šablóna	Zbieranie predmetov
Ovládanie	Relatívny pohyb
Vyhodnocovanie	Postupné
Dátum	28.5.2014
Dĺžka riešenia	2:16
Správnosť riešenia	Správne
Počet pokusov	2
Počet nápovied	0
Riešenie	D,R,R,D,R,D,R,D,

Obr. 27: Náhľad vyexportovaných výsledkov

Záver

Cieľom práce bolo preskúmať dostupné programátorské prostredia a programy na vyučovanie základov programovania a úlohy, ktoré sa v nich používajú. Odskúšali sme päť programovacích prostredí a vyšpecifikovali šesť typov úloh najčastejšie používaných v týchto prostrediach alebo informatických súťažiach. Na základe zistených informácii sme navrhli komplexný systém, v ktorom je možné vytvárať a riešiť podobné úlohy. Časť návrhu sme implementovali a vytvorili tak funkčnú aplikáciu.

Náš softvér, GEVIN, obsahuje vlastnosti, ktoré pomáhajú učiteľovi s generovaním príkladov, hľadaním riešení a zároveň s vyhodnocovaním žiackeho riešenia. Práve týmito vlastnosťami sa odlišuje od iných mini jazykov a programovacích prostredí, čo môže prispieť k jeho atraktivite a odlíšeniu sa od existujúcich nástrojov.

Systém je použiteľný na vyučovanie základov programovania na základných školách. Beta verziu softvéru sme testovali zo žiačkou základnej školy a učiteľkou informatickej výchovy a na základe zistených informácii sme softvér upravili a pridali niekoľko ďalších rozšírení do oboch častí systému.

V rámci tejto práce sme do systému implementovali iba časť z navrhnutých funkcionalít a šablón, ale plánujeme vo vývoji pokračovať. Aktuálnu verziu aplikácie sa chystáme otestovať s väčšou skupinou žiakov a pedagógmi, vyučujúcimi informatickú výchovu na základnej škole. Podľa zistených skutočností budú v systéme upravené zistené nedostatky a zároveň budeme zisťovať ďalšie možnosti rozšírenia.

Veríme, že GEVIN nájde uplatnenie vo vyučovacom procese

Literatúra a internetové zdroje

[1] Pauchly, Karol. (2002) Výučbové programovacie mini-jazyky[online]. [10.2.2014] Dostupné na <http://www.flatulent.szm.com/karel/minijazyky.html>

[2] Baltík 3 [online] [10.2.2014] Dostupné na <http://baltik.infovek.sk/zakl_inf.htm>

[3] Jedlička, Oldřich.Karel,Napověda, [online] [10.2.2014] Dostupné na<http://karel. oldium.net/napoveda.html>

[4] Černík, Andrej. (2008) Stručná informácia o programe. [online] [12.2.2014] Dostupné na <http://www.emil.input.sk/info_sk.htm>

[5] Salanci, Ľubomír. EasyLogo-discovering basic programmin gconcepts in a constructive manner. In: Constructionis tapproaches to creative learning, thinking and education: Lessons for the 21st century. Bratislava: FMFI UK, 2010. ISBN 978-80-89186-6-2, ISBN 978-80-89186-65-5 [online] [12.2.2014] Dostupné na <http://www.salanci.sk/EasyLogo/Paper.pdf>

[6] Lifelong Kindergarten Group, MIT MediaLab, Getting Startedwith Scratch [online] [14.2.2014] dostupné na <http://cdn.scratch.mit.edu/scratchr2/static/1389309042//pdfs/help/Getting-Started-Guide-Scratch2.pdf>

[7] Ondková, Jana. (2006) Detský programovací jazyk Mravec pre 1. stupeň ZŠ, Diplomová práca

[8] KZVI FMFI UK, iBobor – súťaž, [online] [15.2.2014] Dostupné na <http://ibobor.sk/sutaz_demo/index.php>

[9] Pecinovský, Rudolf. (2004). Proč a jak učit OOP žáky základních a stredních škol [online] [18.2.2014] Dostupné na <http://vyuka.pecinovsky.cz/prispevky/2004_ZDK_Proc _a_jak_ucit_OOP_zaky_ZS_a_SS.pdf>

[10] Chiasson, Sonia, Gutwin, Carl (2005). Design principles for children´s technology. [online] [20.2.2014] Dostupné na <http://www.hci.usask.ca/publications/2005/HCI_TR_2005_02_Design.pdf>

[11] Oracle, (1995, 2014), The Java Tutorials, [online] [5.5.2014] Dostupné na <http://docs.oracle.com/javase/tutorial/index.html>

[12] Eclipse, [online] [15.12.2013] Dostupné na <http://www.eclipse.org/>

[13] S. Dasgupta, C. H. Papadimitriou, a U. V. Vazirani, Algorithms, MCGraw-Hill Higher Education 2006,Dostupné na<http://www.cs.berkeley.edu/~vazirani/algorithms.html>

[14] Inkscape, [online] [19.4.2014] Dostupné na <http://www.inkscape.org/cs/>

[15] Y. Shafranovich, (2005), Common Format and MIME Type for Comma-Separated Values (CSV) Files, [online] [20.5.2014] Dostupné na <http://tools.ietf.org/html/rfc4180>

Prílohy

Digitálna príloha – Kompaktný disk s aplikáciou a zdrojovými kódmi