Sem vložte zadání Vaší práce.

České vysoké učení technické v Praze
Fakulta informačních technologií
Katedra Softwarového Inženýrství

Bakalářská práce

Interaktivní kurz o algoritmickém řešení problémů

Martin Kopřiva

Vedoucí práce: Ing. Bc. Ivan Ryant

16. února 2016

Poděkování

Chtěl bych poděkovat Ing. Bc. Ivanu Ryantovi za vedení této práce a za odborné rady a připomínky. Také bych rád poděkoval rodině a přátelům za podporu.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou, a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 16. února 2016 .....

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2016 Martin Kopřiva. Všechna práva vyhrazena.
Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.

Odkaz na tuto práci

Kopřiva, Martin. *Interaktivní kurz o algoritmickém řešení problémů.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tato práce se zabývá vývojem interaktivní výukové videohry zaměřené na téma algoritmizace úloh vyučované na gymnáziích a spustitelné pod operačním systémem Windows a Linux. Na základě analýzy existujících řešení a požadavků je navržena a implementována aplikace. Její součástí je interaktivní herní svět, různé algoritmické úlohy a vývojové prostředí.

Klíčová slova interaktivní výuková videohra, Java, libGDX, JSON, Nashorn, algorithmizace

Abstract

This bachelor thesis deals with development of interactive educational video game aimed at subject of algorithmization of tasks that is taught at grammar schools. This video game is executable under operating system Windows and Linux. The application is designed and implemented on the basis of analysis of existing solutions and requirements. The application includes interactive game world, different algorithmic problems and integrated development enviroment.

Keywords interactive educational video game, Java, libGDX, JSON, Nashorn, algorithmization

Obsah

Úvod **1**

1 Cíl práce **3**

2 Analýza **5**
2.1 Výukový software 5
2.2 Algoritmizace a základy programování 9
2.3 Současný stav řešení problematiky 19
2.4 Možnosti řešení 21
2.5 Zvolené řešení 22
2.6 Volba jazyka a knihoven 23
2.7 Analýza požadavků 23
2.8 Model případů užití 24

3 Návrh **27**
3.1 Grafické uživatelské rozhraní aplikace 27
3.2 Statistiky 29
3.3 Vývojové prostředí 30
3.4 Herní svět 34
3.5 Příběh 37
3.6 Algoritmické úlohy 37

4 Realizace **39**
4.1 Interakce 39
4.2 Vývojového prostředí 39
4.3 Persistence dat 42
4.4 Lokalizace 43
4.5 Zvuk 43
4.6 Technologie a nástroje 43
4.7 Dokumentace 45

4.8 Testování 45

Závěr **47**
Osobní přínos 47
Výhled do budoucna 47

Použité zdroje **49**

A Seznam použitých zkratek **51**

B Obsah přiloženého CD **53**

C Uživatelská příručka **55**
C.1 O aplikaci 55
C.2 Profil hráče 55
C.3 Změna jazyka 55
C.4 Ovládání ve hře 55
C.5 Ovládání postavy 56
C.6 Vývojové prostředí 56

Seznam obrázků

2.1 Stavový diagram 12
2.2 Vývojový diagram 12
2.3 Vývojové prostředí Scratch 20
2.4 Vývojové prostředí BlueJ 21
2.5 Model případů užití 25
2.6 Pokrytí požadavků 26

3.1 Návrh okna nového hráče 28
3.2 Návrh hlavní nabídky 29
3.3 Návrh okna profilu hráče 30
3.4 Návrh okna vývojového prostředí 33
3.5 Návrh okna vývojového prostředí za běhu programu 34
3.6 Návrh herního světa 35
3.7 Návrhový model tříd herního světa 36

4.1 Dialogy kontroly chyb 43

C.1 Ikona interakce 56
C.2 Vývojové prostředí 57
C.3 Požadovaný cíl před a po úspěšném běhu 57
C.4 Zamknutý příkaz 58
C.5 Přesun příkazu 59
C.6 Zamknutá hodnota a vybrání hodnoty 59
C.7 Dialogy kontroly chyb 60

Úvod

Algoritmizace a základy programování jsou na gymnáziích vyučovány již mnoho let. Tato oblast je pro studenty obtížnou především díky odlišnosti od ostatních probíraných předmětů a silné závislosti znalostí a porozumění tématu na praktických zkušenostech, které si musí vlastnoručně osvojit za pomoci procvičování a práce s jednotlivými částmi. V současnosti jsou k těmto účelům využívány různé programy, které však nemusí přímo odpovídat rozsahu látky či jsou pro začátečníky zbytečně složité.

Z tohoto důvodu je potřeba vytvořit výukový program, který bude jednoduchý na ovládání, zaměřený na toto téma, studentům pomůže při překonání počátečních obtíží a názorně ukáže, co mohou za pomoci algoritmizace dokázat.

První kapitola specifikuje cíle, kterých by měla práce dosáhnout.

Druhá kapitola se zabývá pojmem výukový software, algoritmizací a programováním. Dále obsahuje analýzu existujících řešení, výběr vhodného řešení, volbu jazyka a knihoven, analýzu požadavků a model případů užití.

Třetí kapitola pojednává o návrhu jednotlivých částí aplikace, které je potřeba vytvořit.

Čtvrtá kapitola se věnuje podrobnostem realizace aplikace a testování.

V závěru je poté shrnuto splnění požadovaných cílů práce, její stav, osobní přínos a výhled do budoucna.

Kapitola 1

Cíl práce

Cílem této práce je analyzovat, navrhnout a implementovat výukový program, který bude sloužit jako pomůcka při výuce dílčího tématu Algoritmizace úloh obsaženého v RVP-G, oboru Informatika a ICT, tematického okruhu Zpracování a prezentace informací [1]. Dále také zvolit vhodný jazyk a knihovny pro potřebnou funkčnost.
Tento program bude:

- umožňovat žákovi sestavovat a spouštět programy
- umožňovat žákovi sledovat běh programů
- evidovat statistiky o žákově postupu
- umožňovat ukládání a nahrávání postupu žáka
- umožňovat žákovi interagovat s prostředím

Kapitola 2

Analýza

2.1 Výukový software

Výukový software je důležitou součástí moderního vzdělávacího procesu a lze jej v dnešní době nalézt v převážné většině podob vzdělávání. Mezi tyto podoby patří jak samotná výuka ve školách, tak i různé kroužky a volnočasové aktivity. Jde o důležitého pomocníka, jehož cílem je zoptimalizovat, zefektivnit a zjednodušit práci vyučujících, ne však jejich roli ve vzdělávacím procesu zcela nahradit [2]. Často je tedy využíván při průchodu učivem, procvičování či testování.

Samotný výukový software lze definovat podle [2, str. 23] jako „jakékoliv programové vybavení počítače, které je určeno k výukovým účelům a dokáže plnit alespoň některou z didaktických funkcí“. Podle dané definice odpovídá tedy pojem výukový software pojmu didaktický software.

V anglické literatuře je také možné nalézt pojem edukační software, který je však podle [2, str. 24] definován jako „jakékoliv programové vybavení počítače, které je předurčeno pro využití v situacích, kdy dochází k rozvoji osobnosti jedince“.

Jako poslední je třeba jmenovat pojem výukový program, který lze nalézt v české literatuře. Tímto pojmem je podle [2, str. 24] označován „konkrétní software určený k výukovým účelům“.

2.1.1 Kategorizace

Pro lepší pochopení funkčnosti a vlastností daného výukového softwaru je třeba jej podle [2] rozdělit do jednotlivých kategorií na základě následujících kriterií.

1. Podle míry interaktivity:
    - interaktivní
    - bez interaktivních prvků

Důležitou vlastností výukových programů je interaktivita a to předně z didaktického hlediska. Jejím základem je schopnost uživatele ovlivňovat průběh daného kurzu a nebýt tak pouhým pasivním divákem. Jde tedy o vzájemnou komunikaci, při které žák aktivně participuje ve vzdělávacím procesu a je tedy více motivován.

2. Podle úrovně vzdělávání:

   - pro mateřské školy
   - pro základní školy
   - pro střední školy
   - pro vysoké školy

   Výukový software lze v současnosti nalézt ve všech částech vzdělávacího systému. Proto je důležité obsah daného programu zaměřit na specifickou vzdělávanou skupinu a její výchovně-vzdělávací cíle. Jmenované zaměření je kritické pro splnění funkce výukového programu.

3. Podle míry poskytování zpětné vazby:

   - zpětnovazební
   - bez zpětné vazby

   Zpětná vazba je velmi potřebná vlastnost výukových programů, která je často opomíjena. Je zvláště důležitá u programů, které obsahují části zaměřené na cvičení a testovaní osvojených znalostí. Žák je na základě této zpětné vazby informován o kvalitě a správnosti svého dosavadního pochopení potřebných poznatků. Podle získaných informací má pak možnost odstranit případné nedostatky či nepřesnosti.

4. Podle organizovanosti vzdělávání:

   - pro školní výuku
   - pro samostudium

   Výukový program nemusí být využíván pouze ve školním prostředí, ale také jako účinný nástroj při samostudiu, kdy je žák sám zodpovědný za průchod daným výukovým obsahem. Samotný průchod může být asistován daným programem.

5. Podle on-line x off-line funkčnosti:

   - off-line
   - off-line s on-line podporou
   - on-line

Výukový software je možné nalézt v podobě lokální off-line verze, kdy je program nainstalován přímo na počítači či školním serveru, dále pak v podobě on-line, kdy se žák připojuje k danému programu pomocí internetu a jejich kombinaci off-line programu s on-line podporou. Tato podpora může představovat různé aktualizace či speciální online části zaměřené například na testování.

6. Podle počtu uživatelů:

   - monouživatelský
   - víceuživatelský

   Víceuživatelská varianta výukového softwaru je většinou pracnější možností, poskytuje však mnohé přídavné výhody. Mezi tyto výhody lze počítat rozvoj kooperace a sociálních schopností zúčastněných a dále také lepší motivaci žáka při spolupráci s kamarádem. S touto variantou se lze setkat ve dvou podobách. První obsahuje systémy pro spolupráci více uživatelských klientů zároveň za pomoci internetu či lokální sítě. Druhá varianta je vytvořena tak, aby mohlo jednoho uživatelského klienta využívat více žáků zároveň.

7. Podle tematického rozsahu:

   - monotematicky
   - polytematicky

   Výukový software může být zaměřen na jedno či více témat. Jestliže program pokrývá více témat, musí být správně strukturován pro dobrou přehlednost.

8. Podle možnosti vnímání:

   - vizuální
   - audiovizuální

   Výukový program, který obsahuje mimo vizuální stránky ještě stránku zvukovou, může být pro žáka poutavější a může tak zlepšit celkový dojem a tím i nabyté znalosti. Je však nutné uvést, že velké množství školních institucí z různých důvodů, jako je například potřeba vnímání vyučujícího studenty, nedisponuje vybavením pro přehrávání audia.

9. Podle jazykových mutací:

   - jednojazyčný
   - vícejazyčný

Výukový software většinou obsahuje pouze jednu jazykovou stopu. Tato stopa zahrnuje mateřský jazyk žáka, pro kterého je software určen. Existují však i programy mezinárodního charakteru, které buď přímo zahrnují podporu několika jazykových stop, nebo je možné je o dodatečné jazykové lokalizace rozšířit. Tento software lze také využít pro zlepšení jazykových schopností zúčastněných.

10. Podle počtu didaktických funkcí:

    - s jednou didaktickou funkcí (motivační, expoziční, fixační, verifikační)
    - didakticky polyfunkční

    Jak již bylo v této kapitole uvedeno, výukový software musí podle definice pokrývat nejméně jednu didaktickou funkci jako je procvičování, výklad či testování látky. Software, který má více didaktických funkcí je označován termínem didakticky polyfunkční.

11. Podle zaměření na jednotlivé předměty:

    - předmětově zaměřený (informatika, matematika, fyzika...)
    - bez předmětového zaměření

    Většina výukových programů je zaměřena přímo na jeden předmět. Existují však i programy obecnějšího charakteru.

2.1.2 Volba programů pro výuku

Rozhodnutí o zvolení programu pro konkrétní výuku je z mnoha hledisek nelehkou záležitostí. Podle [2] je třeba se rozhodovat s ohledem na následující specifikace:

1. výukové cíle

   Jedná se o primární cíle, které je potřeba s využitím programu splnit.

2. věk a úroveň psychického vývoje žáků

   Nastavení programu pro danou věkovou skupinu je důležitou vlastností, která může ovlivnit efektivitu výuky.

3. schopnosti učitele integrovat je do výuky

   Učitel musí být schopen využít plného potenciálu programu a zároveň také formát a obsah programu musí být vhodný pro daný typ výuky.

4. podmínky realizace

   Mezi podmínky realizace lze počítat dostupnost samotného programu na daném zařízení a dostatečné technické vybavení využívané při výuce.

2.1.3 Didaktická počítačová hra

O hře lze obecně hovořit podle [2, str. 26] jako o činnosti „jednoho nebo více lidí, která nemusí mít konkrétní smysl, ale přitom má za cíl vytvářet radost či působit relaxačně“. Nemělo by však jít pouze o činnost zaměřenou na zábavu, měla by také rozvíjet schopnosti zúčastněných.

S rozvojem techniky přišly hry počítačové, které jsou založené na vlastním virtuálním světě, kterého se hráč stává součástí za pomoci vstupních periferií, které mu umožňují v tomto světě vykonávat různé činnosti a tím jej měnit. Hráčovým cílem je většinou splnit co nejefektivněji či co nejpřesněji předem známé úkoly a tím postoupit dále či získat odměnu.

Samotný virtuální herní svět je nejdůležitější součástí počítačových her a umožňuje hráči prozkoumávat různá nereálná či reálná prostředí. Za pomoci daného světa si hráč může doplnit znalosti o jinak jemu málo dostupných či zcela nedostupných lokalitách. Jako příklad lze uvést hry obsahující simulaci vesmíru či hry vyobrazující různá exotická prostředí a jejich faunu a flóru.

Hry, které jsou zaměřené na výuku, jsou podle [2] nazývány hrami didaktickými. Didaktické počítačové hry jsou důležitou podmnožinou, která je, díky svému výchovně-vzdělávacímu zaměření a virtuálnímu světu, ideální pro rozvoj osobnosti hráče. Účastník si hraje a zároveň nabývá nové poznatky. Hráč tedy podle typu didaktické hry zlepšuje své strategické myšlení, počítačovou gramotnost, sociální dovednosti či tvořivost.

2.2 Algoritmizace a základy programování

Tato sekce je zaměřena na vymezení potřebných pojmů a částí algoritmizace a programování.

2.2.1 Algoritmus

Algoritmus je podle [3, str. 25] „předpis, který se skládá z kroků a který zabezpečí, že na základě vstupních dat jsou poskytnuta požadovaná data výstupní“. Musí také podle [3] splňovat následující vlastnosti:

1. Konečnost

   Předpokládaného cíle je třeba dosáhnout v konečném počtu kroků. Tento počet by měl být dostatečně malý, aby výsledek práce algoritmu bylo možné využít. Procedura, která splňuje všechny potřebné předpoklady algoritmu, ale není konečného charakteru, je podle [4] nazývána výpočetní metodou. Jako příklad neukončené výpočetní metody lze jmenovat reaktivní proces, který reaguje na vnější podněty.

2. Hromadnost

   Algoritmus by mělo být možné využít pro řešení většího množství stejného typu problémů. Tato vlastnost by měla být zajištěna variabilitou vstupních dat algoritmu. Při vytváření textové podoby algoritmu je variability dosaženo označením vstupů pojmenovanými proměnnými.

3. Jednoznačnost

   Všechny kroky algoritmu musí být přesně definovány. Pro každý stav tedy musí být jednoznačné, který krok bude následovat.

4. Opakovatelnost

   Algoritmus by měl na základě vstupů stejných hodnot dosáhnout stejných výsledků.

5. Rezultativnost

   Použití algoritmu by mělo vést k výsledku. Tento výsledek je reprezentován výstupy.

2.2.2 Vytváření algoritmu

Při vytváření algoritmů bývají dodržovány základní zásady a využity osvědčené metody.

Využívanými metodami jsou:

1. Metoda shora dolů

   Základem je rozložení daného problému na části neboli podproblémy, které lze, pokud je to potřeba, dále rozdělit. Výsledkem opakování rozkladu je pak posloupnost elementárních prvků.

2. Metoda zdola nahoru

   Algoritmus je vytvářen od jednotlivých elementárních prvků, které jsou pak skládány do složitějších struktur.

2.2.3 Zápis algoritmu

Algoritmy je možné popsat množstvím různých způsobů. Tyto způsoby jsou založeny na grafické či slovní bázi. Mezi způsoby je třeba vybírat na základě dané úlohy či zkušeností programátora. Jakožto zástupce používaných způsobů lze jmenovat:

1. Využití jazyka pro popis programů

   Jde o velmi často používaný způsob, při kterém je algoritmus popsán za pomoci slovního zápisu. Tento zápis se skládá ze strukturovaného

rozkladu algoritmu na jednotlivé jednoduché kroky vytvořeného za pomoci metody shora dolů. Některé typy zápisu je následně možné využít při okomentování výsledného programu.

Tento způsob popisu lze rozdělit do kategorií podle použitého jazyka:

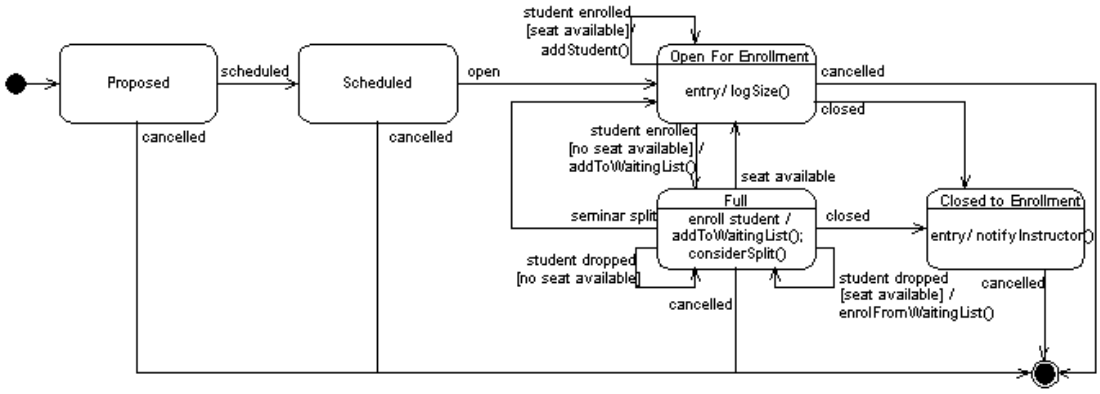
- Využívající přirozený jazyk
  Tato kategorie zahrnuje především popis algoritmu vyjádřený tak, jak bychom jej popsali při obyčejné konverzaci. Problémem takového popisu je častá nejednoznačnost posloupnosti jednotlivých kroků způsobená využitými slovními obraty a větnou strukturou. Na druhou stranu je však tento způsob často jednoduchý na pochopení.
- Využívající speciální jazyk
  Do této kategorie připadá převážně pseudokód, který je většinou založen na konvencích programovacích jazyků. Mezi tyto konvence patří především strukturovanost samotného zápisu. Hlavní výhodou využití pseudokódu oproti programovacímu jazyku je absence pro popis algoritmu zbytečných částí syntaxe, která většinou zaručuje lepší čitelnost a srozumitelnost. Jde tedy o vhodný způsob zápisu pro začínající programátory.
- Využívající programovací jazyk
  Výsledný zápis je většinou možné přímo využít v právě vytvářeném programu. Zápis však často vyžaduje již pokročilou znalost daného jazyka. Jeho srozumitelnost je závislá na kvalitě označení a pojmenování jednotlivých důležitých částí a využitém programovacím jazyce.

2. Stavový diagram

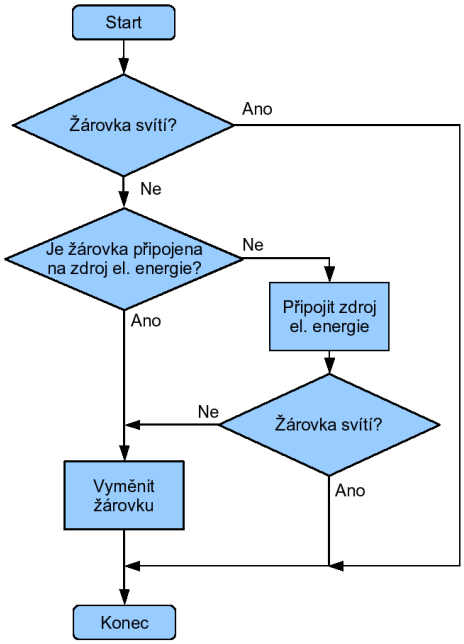
   Stavový diagram je způsobem grafického zápisu v UML. Je složen z jednotlivých stavů, přechodů a pseudostavů. Stavy jsou znázorněny obdélníky se zaoblenými rohy. Šipky představují přechody a jejich směr. Tyto přechody zahrnují jednu nebo více událostí a mohou také obsahovat podmínku a akci. Mezi pseudostavy patří počátek, konec, větvení apod. Stavový diagram je znázorněn na obrázku 2.1.

3. Vývojový diagram

   Vývojový diagram je klasickým způsobem grafického zápisu algoritmu. Jeho kroky jsou popsány za pomoci různých geometrických tvarů označující typ daného příkazu, obsahující slovní popis dané činnosti či rozhodování. Jednotlivé kroky jsou spojeny za pomoci šipek, které označují směr přechodu mezi nimi viz obrázek 2.2. Tento způsob je v současnosti, díky své náročnosti na samotnou výrobu a následné úpravy, využíván zřídka.



Obrázek 2.1: Příklad stavového diagramu



Obrázek 2.2: Příklad vývojového diagramu

2.2.4 Jednoduché datové typy

Datové typy jsou určeny množinou hodnot a operacemi nad nimi. Mezi jednoduché datové typy patří:

- Logické

  Bývají označovány slovem bool či boolean. Mohou nabývat hodnot true (pravda) a false (nepravda).

- Celočíselné

  Bývají označovány slovem integer. Nabývají hodnot z množiny celých čísel. Rozsah prvků této množiny je závislý na konkrétní implementaci.

- Reálné

  Bývají označovány slovem real, double či float. Nabývají hodnot z množiny reálných čísel. Rozsah prvků této množiny a přesnost reprezentace je závislá na konkrétní implementaci.

- Znakové

  Bývají označovány slovem char. Množina jejich hodnot se skládá z číslic, znaků abecedy a speciálních znaků. Každé hodnotě je přidělena celočíselná hodnota, která je označována jako kód znaku. Pro kódování znaků je využíváno znakových sad, jako je například ASCII.

2.2.5 Proměnná

Proměnná je pojmenované místo v paměti počítače. Součástí deklarace je identifikátor a typ. Typ proměnné označuje množinu možných hodnot. Identifikátor je využíván pro přístup a operace s proměnnou. Může tedy měnit svou hodnotu.

Proměnné jsou děleny podle způsobu alokace na:

- Globální proměnné

  Jsou vytvořeny při spuštění programu a existují po celou dobu běhu programu.

- Lokální proměnné

  Jsou vytvořeny v okamžiku volání funkce a při jejím dokončení zanikají. Pro každou instanci jsou vytvořeny nové lokální proměnné.

- Dynamické proměnné

  Jsou vytvořeny zavoláním příkazu pro alokaci paměti a zanikají za použití speciálního příkazu pro uvolnění paměti či za pomoci automatizovaných systémů správy paměti.

2.2.6 Konstanta

Podobně jako proměnná má identifikátor a datový typ. Její hodnota se však nemění.

2.2.7 Základní datové struktury

Datové struktury definují způsob organizace dat v paměti a většinou také operace nad nimi. Za základní datové struktury lze označovat pole, záznam a objekt.

2.2.7.1 Pole

Jde o posloupnost proměnných stejného typu. Prvky pole jsou přístupné přímo za pomoci indexů. Tyto prvky jsou v paměti uloženy za sebou. Deklarace pole je, oproti proměnné, rozšířena o číslo udávající počet prvků. Pole jsou dělena podle obsahu na dva typy:

- jednorozměrné

  Je základním typem pole mezi jehož prvky nepatří další pole. Pro přístup k prvkům stačí jeden index.

- vícerozměrné

  Jde většinou o pole, které je podobné jednorozměrnému. Jeho prvky jsou však pole délky odpovídající velikosti daného rozměru. V paměti jsou pak složky tohoto pole uloženy tak, že se nejrychleji mění poslední index [5]. U programovacích jazyků, které tento typ nepodporují přímo, jde také o pole, jehož prvky jsou další pole. Na rozdíl od předchozí varianty však nemusí tyto prvky mít stejnou délku podle daného rozměru.

2.2.7.2 Záznam

Jde o skupinu proměnných, která je seskupena pod jedním identifikátorem. Pro přístup do jednotlivých položek je třeba uvést společně s identifikátorem záznamu i jméno dané položky. Tento způsob přístupu je označován kvalifikace.

2.2.7.3 Objekt

Objekt je základem objektově orientovaného programování a příslušných programovacích jazyků. Objekt bývá založen na předpisu objektového typu neboli třídy.

Definice objektového typu obsahuje nejen atributy, které jsou datovými složkami, ale také metody. Samotná instance neboli objekt daného typu pak obsahuje pouze atributy. Tyto atributy mohou být soukromé a nelze k nim

tedy přistupovat přímo či vůbec. Atributy třídy jsou v podstatě globální proměnné, které jsou k této třídě přidružené.

2.2.8 Odvozené datové struktury

Bývají také označovány jako abstraktní datové struktury. Tohoto označení je však většinou využíváno při realizaci těchto struktur za pomoci objektů [5].

2.2.8.1 Seznam

Seznam je rekurzivní datovou strukturou. Jde o posloupnost prvků stejného typu uspořádaných podle pořadí, v kterém byly přidány, či určitého klíče. Tímto klíčem může být hodnota dat obsažených v prvcích či hodnota funkce, která s těmito daty pracuje. Tyto prvky obsahují jeden nebo více odkazů na ostatní prvky v seznamu. Pomocí odkazů je možné seznam projít. První prvek je nazýván hlava neboli head a pomocí odkazu na něj lze k obsahu seznamu přistupovat. Nelze tedy přímo přistupovat k jakémukoliv prvku.

2.2.8.2 Kořenový strom

Jde o rekurzivní datovou strukturu. Prvky stromu jsou označovány jako uzly. Tyto prvky mohou mít jednoho rodiče a potomky. Uzel, který nemá rodiče je kořenem. Uzel, který nemá potomky se nazývá list. Samotný strom obsahuje maximálně jeden kořen, na který lze přímo přistoupit, a od kterého je možné projít celý strom. Jsou využívány pro uchovávání hierarchicky uspořádaných dat. Stromy není pro potřeby práce nutné více rozvíjet.

2.2.8.3 Zásobník

Jde o strukturu, u které lze interagovat pouze s prvkem, který byl přidán jako poslední neboli je na vrchol zásobníku. Tato vlastnost je označována pojmem LIFO a je jí využíváno pokud je potřeba pracovat s prvky v opačném pořadí, než v jakém byly přidány.

Nad touto strukturou jsou specifikovány operace:

- push

  Přidání prvku na vrchol zásobníku.

- pop

  Odebrání prvku z vrcholu zásobníku.

- top

  Dotaz na vrchol zásobníku.

- isEmpty

  Jde o test prázdnosti zásobníku.

2.2.8.4 Fronta

Do fronty jsou prvky přidávány na jedné straně, kterou je konec fronty. Oproti zásobníku jsou však prvky odebírány z druhé strany neboli začátku fronty. Jde tedy o FIFO datovou strukturu.

Nad touto strukturou jsou specifikovány operace:

- enqueue

  Přidání prvku na konec fronty.

- dequeue

  Odebrání prvku ze začátku fronty.

- isEmpty

  Jde o test prázdnosti fronty.

2.2.9 Funkce

Je částí programu, kterou je možné v programu opakovaně volat. Může mít parametry neboli vstupní hodnoty a návratovou hodnotu.

2.2.10 Rekurze

Rekurze je důležitou technikou využívanou při algoritmizaci a programování. Její definující vlastností je opakované volání stejné funkce. K samotnému volání dochází v jejím těle. Výsledkem této činnosti je postupné rozdělení původní úlohy na menší podúlohy.

Rekurze je dělena podle volané funkce na dva druhy:

- přímá

  Při této rekurzi dochází v těle funkce k jejímu volání. Tato skutečnost je také označována jako volání sebe sama.

- nepřímá

  Je uskutečněna, jestliže funkce volá jinou proceduru. Ta může následně volat další či funkci původní. Funkce tedy při volání sebe sama využívá prostředníka.

Jednotlivá volání v rámci rekurze znamenají alokaci paměti pro každou instanci funkce. Alokace pro instanci zahrnuje jak předané parametry, tak lokální proměnné i návratovou adresu. Při špatném využití či navržení rekurze tedy může dojít k vyčerpání paměti a chybovému ukončení programu. Z tohoto důvodu není rekurze vhodným řešení některých problémů.

2.2.11 Řazení

Řazení lze podle [5] definovat jako „uspořádání zadané posloupnosti dat podle určitého klíče v neklesajícím nebo nerostoucím pořadí“.

Metody řadění jsou rozděleny podle situace na:

- vnitřní řazení

  K vnitřnímu řazení dochází pokud je počet prvků posloupnosti předem známý, všechny prvky řazené posloupnosti jsou uloženy ve vnitřní paměti a lze k nim přistupovat v libovolném pořadí.

- vnější řazení

  K vnějšímu řazení dochází pokud není počet prvků předem známý, prvky řazené posloupnosti jsou ve vnější paměti a lze k nim přistupovat sekvenčně.

Pro účely práce je třeba jmenovat jen jednoho zástupce.

2.2.11.1 Bublinkové řazení

Bublinkové řazení neboli bubblesort je jedním ze způsobů vnitřního řazení. Základem algoritmu je postupné porovnávání dvou sousedících prvků řazené posloupnosti. Jestliže tato dvojice není uspořádána, potom si tyto prvky vymění pozici. Dochází tedy k probublání maximálního prvku posloupnosti na nejvyšší pozici.

2.2.12 Řídící struktury

Řídící struktury rozhodují o dalším průběhu programu. Za jejich pomoci dochází k větvení, opakování a jiným změnám běhu programu.

Základními řídícími strukturami jsou:

1. Sekvence

   Skládá se z posloupnosti jednoho nebo více kroků, které jsou provedeny v přesně daném pořadí a postupně, tedy následující krok navazuje po ukončení předchozího. Tyto kroky nemusí sestávat pouze ze základních příkazů, ale mohou se rozvíjet do dalších sekvencí, cyklů či selekcí.

2. Cyklus

   Je částí programu, která je na základě řídící podmínky opakována. Samotný cyklus je tedy složen z řídící podmínky a těla cyklu. Tělem cyklu jsou nazývány příkazy, které se opakují.

   Cykly se dělí do dvou typů:

- indukční

  Tělo cyklu je opakováno na základě řídící podmínky. Tato podmínka tedy rozhoduje, bude-li posloupnost obsažená v těle provedena, nebo dojde k ukončení a přechodu k dalšímu kroku.

- iterační

  Tělo cyklu je opakováno na základě hodnoty řídící proměnné.

Cykly jsou také rozděleny podle umístění podmínky do následujících druhů:

- cyklus s podmínkou před vykonáním těla cyklu

  Cyklus této podoby bývá označován slovem while nebo for. Cyklus je ukončen, jestliže podmínka není splněna. Může tedy nastat situace, že posloupnost v těle cyklu není vůbec provedena.

- cyklus s podmínkou za tělem cyklu

  Tento druh bývá označován spojením do-while. Splnění podmínky je prověřeno po provedení těla cyklu. Posloupnost je tedy provedena alespoň jednou.

- cyklus s podmínkou uvnitř těla cyklu

  Tento druh bývá modifikací předchozích druhů. Často je řídící podmínka permanentně splněna a tělo cyklu obsahuje podmíněné přerušení většinou využívající příkazu break.

3. Selekce

   Za pomoci selekce neboli podmíněné operace dochází k větvení. Samotná selekce je tvořena podmínkou a jednou či více posloupnostmi příkazů. Z těchto posloupností je na základě podmínky vybrána množina, která je následně provedena. Tato množina může být prázdná.

2.2.13 Etapy řešení problému

Proces řešení problému lze rozdělit podle [3] na následující etapy:

1. specifikace problému

   Je specifikována podstata problému a účel jeho řešení.

2. analýza problému

   Je ověřena řešitelnost problému a zaměřena pozornost na možná řešení. Je vybráno vhodné řešení podle požadovaných vlastností.

3. sestavení algoritmu

   Je sestaven jednoznačný postup z jednotlivých příkazů za jejichž pomoci lze danou úlohu vyřešit.

4. kódování

   Podle vytvořeného algoritmu je sestaven zdrojový kód programu ve vybraném programovacím jazyku.

5. testování

   Jsou odstraňovány chyby obsažené v programu. Prvním typem jsou chyby syntaktické. Jejich odhalení je zajištěno překladačem a na základě jeho výstupu jsou pak odstraněny programátorem. Druhým typem jsou chyby logické, které se mohou projevit za běhu programu, a to nesprávnou činností a chybnými výsledky. Pro jejich identifikaci je využíváno ladícího programu známého pod názvem debugger. Při ladění je využito například sledování aktuálního stavu proměnných a krokování včetně zastavení programu na kritických místech.

2.3 Současný stav řešení problematiky

2.3.1 Výuka

Algoritmizace a základy programování jsou ve světě vyučovány již několik desítek let. Za tuto dobu vzniklo mnoho různých výukových kurzů realizovaných pomocí prezentací, internetových stránek či spustitelných programů, které jsou založeny, na běžně používaných programovacích jazycích, tak i jazycích specificky vytvořených pro výuku.

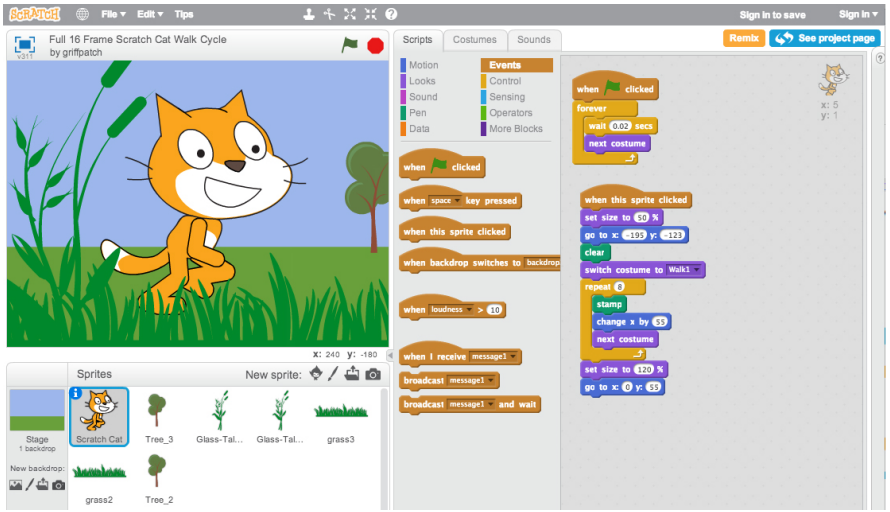
2.3.2 Scratch

V současnosti nejznámějším výukovým programem, který je třeba jmenovat, je Scratch. Jde o vývojové prostředí s vlastním programovacím jazykem, vyvinuté skupinou Lifelong Kindergarten na univerzitě MIT. V tomto grafickém vývojovém prostředí lze pomocí jednoduchých grafických prvků představujících jednotlivé příkazy sestavit program. Hlavní schopností těchto programů je možnost ovládat grafické objekty umístěné na scéně za jejich běhu. Výsledkem jsou uživatelsky vytvořené animace, prezentace a triviální hry. Ukázku vývojového prostředí lze shlédnout na obrázku 2.3.

V tomto vývojovém prostředí však nelze jednoduše využívat některých základních prvků algoritmizace v současnosti používaných v praxi. Mezi tyto prvky patří například plnohodnotně realizované funkce, které obsahuje v dnešní době, mimo triviálních výjimek, každý program. Toto má za následek také obtížnou realizaci důležité programovací techniky jménem rekurze [6].

2.3.3 BlueJ

Jde o vývojové prostředí zaměřené na výuku objektově orientovaného programování v Javě. Za pomoci zjednodušeného UML je možné v tomto prostředí



Obrázek 2.3: Vývojové prostředí Scratch

vytvářet strukturu tříd viz 2.4. Zdrojový kód těchto tříd je pak možné modifikovat za pomoci vestavěného specializovaného textového editoru.

BlueJ bývá využíván při výuce začátečníků založené na přístupu "Objects First", kdy je začátečníkům jako první představeno objektově orientované programování. Jelikož BlueJ využívá jazyka Java, uživatel je nucen se naučit plnou syntaxi tohoto jazyka, která může být pro úplného začátečníka obtížnější na pochopení než zápis za pomoci zjednodušené syntaxe.

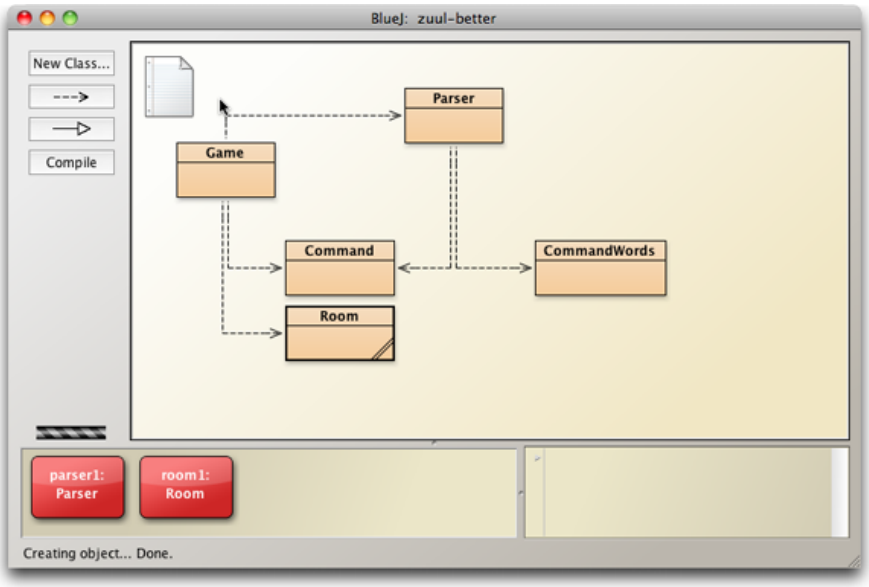
2.3.4 Online kurzy

Dalšími v pořadí jsou výukové online kurzy, kterých je nespočetné množství. Jsou zaměřeny jak na všechny možné programovací jazyky, tak i jako kurzy programovaní samotného. Tyto kurzy obsahují sbírku příkladů, která je většinou rozšířená o motivační příklady a důležité informace potřebné pro jejich splnění. Tyto příklady si může uživatel u lepších kurzů sám pomocí vestavěného jednoduchého vývojového prostředí vyzkoušet a postupovat dále.

Většinou však uživatel není nijak kontrolován, což může být příjemným a užitečným aspektem při samostatném mimoškolním vzdělávání, kdy má účastník zájem se naučit obsah daného kurzu o své vlastní vůli. Nelze však této skutečnosti využít při výuce, kdy je potřeba žáka ohodnotit na základě splněné práce a hlavním cílem je, aby důkladně prošel každou důležitou částí a odnesl si tak potřebné znalosti.

2.3.5 Prezentace

Prezentace je realizovaná pomocí strukturovaných snímků či videa. Jde o vhodnou metodu pro počáteční výklad základních pojmů a prvků využívaných při



Obrázek 2.4: Vývojové prostředí BlueJ

algoritmizaci a programování. Nemůže však být jediným prostředkem výuky. Nelze tak učinit hlavně z hlediska silné závislosti znalostí a porozumění danému tématu na praktických zkušenostech, které si žák musí vlastnoručně osvojit řešením různých algoritmických úloh.

2.4 Možnosti řešení

2.4.1 Platforma

Pro splnění zadání je potřeba, aby vytvořený program fungoval pod operačním systémem Windows a byl přenositelný na systém Linux. Hlavním důvodem je převaha jmenovaných operačních systémů v počítačových učebnách. Tento požadavek částečně omezuje možná řešení o ta, která jsou závislá na platformě, či nefungují konzistentně na různých systémech.

2.4.2 Způsoby řešení

Jak již bylo zmíněno v předchozí části, existuje mnoho různých podob výukových kurzů. Tato řešení sice neodpovídají cílům práce, některé však nejsou příliš vzdálená splnění všech potřebných funkcionalit. Zadání lze splnit vytvořením modifikované verze existujícího řešení či nového programu kombinujícího části z více řešení.

2.4.2.1 Internetový kurz

Základem je soubor internetových stránek nasaditelný na školní server. Tyto stránky, podobně jako v předešlé kapitole, obsahují zadání jednotlivých úloh s potřebnými informacemi uvedenými v co nejsrozumitelnější podobě a zakomponovaným vývojovým prostředím. Tyto úlohy jsou seřazeny tak, aby žáka postupně seznámily se vším potřebným a neobsahovaly nepřekonatelný skok v obtížnosti. Obsahují také systém zajišťující hodnocení, které může vyučující nastavit podle své potřeby a následně kontrolovat, jak si konkrétní žák vede. Výhodou tohoto řešení je možnost využití bez potřeby instalace na jednotlivé počítače. Případným ztížením je však potřeba vytvoření databáze profilů, které by zahrnovaly ukládání hodnocení a rozpracovaných řešení. Částečnou nevýhodou jsou také omezení webové platformy.

2.4.2.2 Výukový herní program

Program je postaven na interaktivním herním světě, díky kterému má žák lepší představu o tom, co přesně algoritmy, které řeší a realizuje, provádějí [7]. Tento herní svět je sestaven z jednotlivých úrovní obsahujících objekty, s kterými může hráč interagovat, a dále také objekty, které následně postupně realizují příkazy uvedené ve vestavěném vývojovém prostředí. Tyto příkazy jsou určeny dílčími algoritmickými úlohami. Nevýhodami takto realizovaného programu je potřeba přídavných systémů realizujících interaktivní herní svět, které je třeba vytvořit oproti ostatním možnostem, a potřeba distribuce či instalace na počítače v učebnách. Výhody, které však toto řešení přináší, značně převažují nevýhody.

2.5 Zvolené řešení

Jakožto řešení, které nejlépe splňuje požadované funkce a atributy, mezi které patří interaktivnost, zábavnost, jednoduchost a efektivnost, autor zvolil interaktivní herní výukový program neboli didaktickou hru ovládanou myší a klávesnicí, obsahující 2D herní svět zobrazený shora.

V tomto světě se může žák, reprezentovaný ovladatelnou postavou, v rámci jednotlivých výukových částí pohybovat a interagovat s objekty. Některé z těchto objektů dávají hráči přístup k zjednodušenému vývojovému prostředí, které je zabudováno přímo do herního programu. V tomto grafickém prostředí může žák jednotlivé části programu, reprezentované grafickými bloky, v rámci dané úlohy přesouvat, přidávat a měnit jejich hodnoty. Výsledný program následně může pomocí příslušného tlačítka, zakomponovaného do vývojového prostředí, spustit.

2.6 Volba jazyka a knihoven

Pro potřeby práce bylo nutné zvolit programovací jazyk a příslušné knihovny, s pomocí kterých lze vytvořit aplikaci s potřebnou funkčností a kompatibilitou s operačním systémem Windows a Linux. Po zvážení dostupných možností, se autor rozhodl pro zvolení programovacího jazyka Java, frameworku libGDX a javascriptového enginu Nashorn.

2.6.1 libGDX

LibGDX je framework pro multiplatformní herní vývoj v Javě. Za pomoci tohoto frameworku lze vytvořit 2D a 3D hry. Součástí je především fyzikální knihovna `box2D`, knihovna `scene2d` zaměřená na grafické uživatelské rozhraní a knihovna JSON, které lze využít pro persistenci dat.

2.6.2 Nashorn

Součástí distribuce Javy od verze 8 je javascriptový engine Nashorn, který lze využít pro interpretaci uživatelem vytvořených programů. Tento engine umožňuje také komunikaci

2.7 Analýza požadavků

2.7.1 Funkční požadavky

- Sestavení uživatelského programu

  Aplikace bude umožňovat sestavení uživatelského programu z jednotlivých dostupných příkazů. Bude také umožňovat měnit hodnoty těchto příkazů.

- Spuštění uživatelského programu

  Aplikace bude umožňovat spuštění a běh programu. Součástí budou také opatření proti syntaktickým a logickým chybám. Při běhu budou zobrazeny a zvýrazněny uživateli užitečné údaje.

- Persistence dat

  Aplikace bude ukládat data potřebná pro navázání na předchozí stav průchodu obsahem kurzu. Budou také ukládána data profilu hráče.

- Interaktivní prostředí

  Aplikace bude umožňovat interakci uživatele s herním prostředím. Uživatel bude interagovat skrze postavu s objekty, které tuto interakci umožňují.

2.7.2 Nefunkční požadavky

- Kompatibilita s OS Windows a Linux

  Aplikaci bude možné spustit pod operačními systémy Windows a Linux.

- Evidence statistik žáka

  Aplikace bude zaznamenávat různé statistiky o postupu žáka. Mezi tyto statistiky budou patřit údaje o jednotlivých úlohách a úrovních. Na základě těchto statistik bude poté vyučujícím vytvářeno hodnocení.

- Herní svět

  Aplikace bude obsahovat 2D animovaný interaktivní svět. Hráč bude v tomto světě reprezentován vlastní postavou, kterou bude schopen ovládat.

- Lokalizace

  Aplikace bude lokalizována v českém a anglickém jazyce.

2.8 Model případů užití

Na základě požadavků byl v UML vytvořen model případů užití viz obrázek 2.5. Tyto případy užití pokrývají jednotlivé funkční požadavky, jak je zvýrazněno na obrázku 2.6.

2.8.1 Případy užití

- Změna hodnoty prvku

  Uživatel může měnit hodnoty prvků programu.

- Přesun prvku

  Uživatel může přesouvat prvky v rámci programu.

- Spuštění programu

  Uživatel může výsledný program spustit. Součástí je také verifikace programu neboli kontrola programu před spuštěním na případné chyby a kontrola chyb za běhu. O případných chybách je uživatel informován.

- Sledování běhu programu

  Uživatel může sledovat běh spuštěného programu.

- Uložení hry

  Uživatel může uložit svůj postup do souboru.

Obrázek 2.5: Model případů užití

- Nahrání hry

  Uživatel může nahrát svůj postup ze souboru.

- Interakce s objektem

  Uživatel může skrze svou postavu interagovat s objekty.

Source \ Target	F1 - Sestavení uživatelského programu	F2 - Spuštění uživatelského programu	F3 - Persistence dat	F4 - Interaktivní prostředí	N1 - Kompatibilita s OS Windows a Linux	N2 - Evidence statistik žáka	N3 - Herní svět	N4 - Lokalizace
Interakce s objektem				↑				
Nahrání hry			↑					
Přesun prvku	↑							
Sledování běhu programu		↑						
Spuštění programu		↑						
Uložení hry			↑					
Verifikace programu		↑						
Změna hodnoty prvku	↑							

Obrázek 2.6: Pokrytí požadavků

Kapitola 3

Návrh

3.1 Grafické uživatelské rozhraní aplikace

Grafické uživatelské rozhraní aplikace je tvořeno jednotlivými okny.

3.1.1 Okno nového hráče

Jde o okno, kterým je hráč přivítán při prvním spuštění aplikace. Obsahuje vstupní textové pole pro jméno hráče, které musí hráč vyplnit řetězcem délky alespoň dvou znaků. Tato podmínka je zkontrolována při stisku tlačítka Vytvořit. Pokud není podmínka splněna, je uživateli zobrazeno upozornění. Jestliže je vše v pořádku, hráčovi se vytvoří profil a aplikace přechází na okno hlavní nabídka. Okno také obsahuje tlačítko Konec, po jehož stisknutí je aplikace ukončena. Návrh okna je zobrazen na obrázku 3.1

3.1.2 Hlavní nabídka

Hlavní nabídka slouží uživateli jako rozcestník pro přístup do jednotlivých částí. Je složena z loga aplikace a seznamu tlačítek viz 3.2. Mezi tato tlačítka patří:

- nová hra

  Stisknutím tlačítka se nahraje úvodní úroveň hry a aplikace přechází na herní okno.

- načíst hru

  Stisknutím tlačítka se nahraje uložená pozice a aplikace přechází na herní okno.

- nastavení

  Stisknutím tlačítka přejde aplikace na okno nastavení.

Obrázek 3.1: Návrh okna nového hráče

- profil hráče

  Stisknutím tlačítka přejde aplikace na okno profilu hráče.

- konec

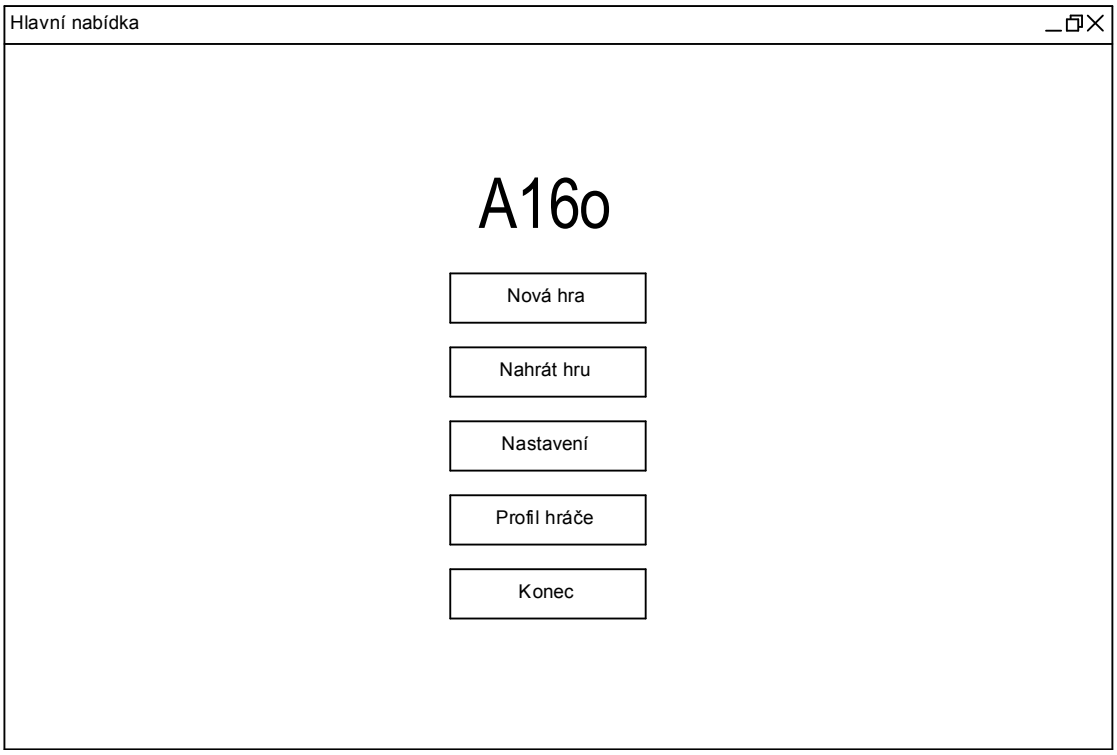
  Stisknutím tlačítka se aplikace ukončí.

3.1.3 Okno nastavení

Toto okno obsahuje tlačítko pro změnu jazyka a tlačítko zpět, pomocí kterého se uživatel dostane do hlavní nabídky.

3.1.4 Okno profilu hráče

V tomto okně lze nalézt jméno hráče, seznam statistik a tlačítko zpět, kterým se uživatel přesune do hlavní nabídky viz obrázek 3.3.



Obrázek 3.2: Návrh hlavní nabídky

3.1.5 Herní okno

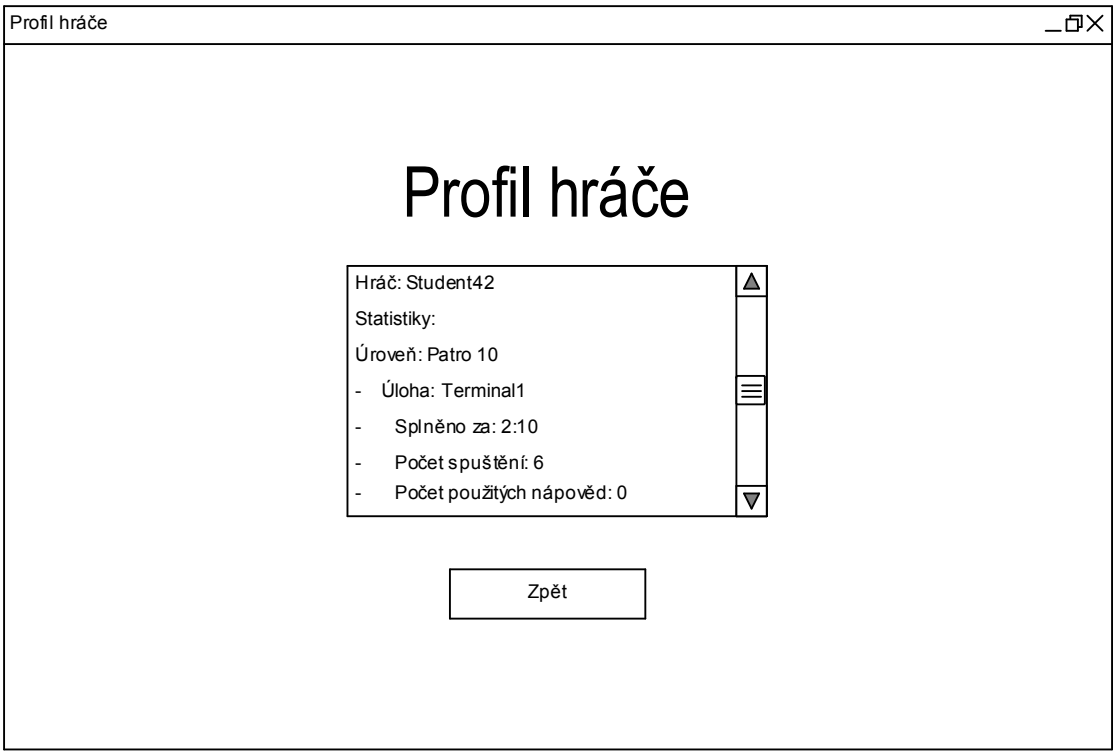
Herní okno zobrazuje aktuálně viditelnou část herního světa, která je snímána kamerou shora. Tato kamera je centrována nad hráčovou postavou. Dále také zahrnuje různá okna uživatelského rozhraní.

3.2 Statistiky

Aplikace obsahuje statistiky, na základě kterých může vyučující následně založit své hodnocení postupu kurzem jednotlivých žáků.

Sledovanými statistikami vyřešených úloh jsou:

- časový interval, za který žák vyřešil danou úlohu
- počet využitých nápověd
- počet spuštění programu, včetně úspěšného běhu



Obrázek 3.3: Návrh okna profilu hráče

3.3 Vývojové prostředí

Jde o nejdůležitější komponentu celé aplikace. Hráč ji využívá při průchodu kurzem a jejím prostřednictvím sestavuje programy, řešící dané algoritmické úlohy, které v něm může také spustit a sledovat jejich průběh. Každá úloha má vlastní instanci prostředí.

3.3.1 Jednoduchost

Hlavní myšlenkou vývojového prostředí je zjednodušit akt programování a usnadní pochopení jednotlivých částí. Zjednodušení je zaměřeno jak na ovládání prostředí tak i na samotný obsah. Prostředí je proto ovládáno pouze myší a množství typů příkazů je limitováno pouze na ty, které jsou důležité pro pochopení základních částí programování. Dále jsou tyto příkazy často omezeny tak, aby uživatele zbytečně nezatěžovaly přílišným množstvím možností, které začátečník ani nevyužije.

3.3.2 Korektnost

Další myšlenkou je také zaručení korektnosti programu vytvořeného ve vývojovém prostředí. Základem je kontrola uživatelem vytvořeného kódu na chyby a limitování možnosti chybu vytvořit. Kontrola syntaktických chyb je provedena před spuštěním a zaměřuje se na hodnoty `null`. Kontrola logických chyb je založena na vkládání přídavných částí kódu do míst, kde tyto chyby mohou vzniknout. Více o kontrole chyb lze nalézt v podsekci 4.2.3. Limitace možností vytváření chyb je realizována tím, že uživatel příkazy nepíše a jejich hodnoty vybírá ze seznamu dostupných možností.

3.3.3 Příkazy

Jsou definovány jednotlivými třídami. Jejich součástí je metoda, která vytvoří javascriptovou reprezentaci daného příkazu.

V rámci kurzu jsou uživateli dostupné následující typy příkazů:

1. deklarace proměnné

   Sestává z předdefinovaného jména, datového typu a její součástí je také přiřazení hodnoty.

2. deklarace pole

   Sestává z předdefinovaného jména, datového typu a její součástí je také přiřazení hodnot.

3. cyklus

   Jsou dostupné všechny tři varianty:

   - for cyklus

     Ze začátku kurzu uživatel využívá zjednodušené verze, kde může pouze změnit počet opakování. Později je mu zpřístupněna plná funkčnost.

   - while cyklus a do-while cyklus

     Obsahuje podmínku s maximálně dvěma členy a operátorem.

4. přiřazení s výrazem

   Tento příkaz lze využít buď jako samotné přiřazení nebo jej rozšířit o výraz, který může obsahovat maximálně dva členy a operátor.

5. podmínka

   Může obsahovat maximálně dva členy a operátor.

6. funkce

   Obsahuje jméno, maximálně dva argumenty a návratovou hodnotu.

7. volání funkce

8. volání metody objektu

   Je využíváno místo klasického vstupu a výstupu. Pomocí těchto metod program komunikuje s herním světem. Součástí volání může být i přiřazení návratové hodnoty do proměnné.

Vývojové prostředí je navrženo tak, aby jej uživatel ovládal výhradně za pomoci myši. Je tak učiněno pro zjednodušení práce s prostředím a usnadnění samotného aktu vytváření programu. Z tohoto důvodu jsou také jednotlivé příkazy zobrazovány uživateli v pseudokódu. Umožňují tak začátečníkovi snadnější pochopení jejich funkce.

3.3.4 Model-view-presenter

Návrh vývojového prostředí je založen na architektonickém vzoru Model-view-presenter [8]. Vývojové prostředí je tedy rozděleno na model reprezentovaný jádrem, presenter a view, kterým je grafické okno. Komunikace mezi jádrem a oknem je zprostředkována skrze presentera. Jádro nekomunikuje s presenterem přímo, ale vysílá při jakékoliv změně události, které presenter odebírá a podle předané události provádí změny v okně. Podobně okno při provedení změn či pokusu o přesun prvku informuje presentera, který následně spustí metody v jádru. Výhodou této varianty je oddělení prezentační části od doménové části. Dále také možnost vytvoření více různých oken nad jednou instancí jádra, která jsou zcela synchronizována s jádrem. Provedení změny v jádře jedním z oken se tedy projeví ve všech ostatních.

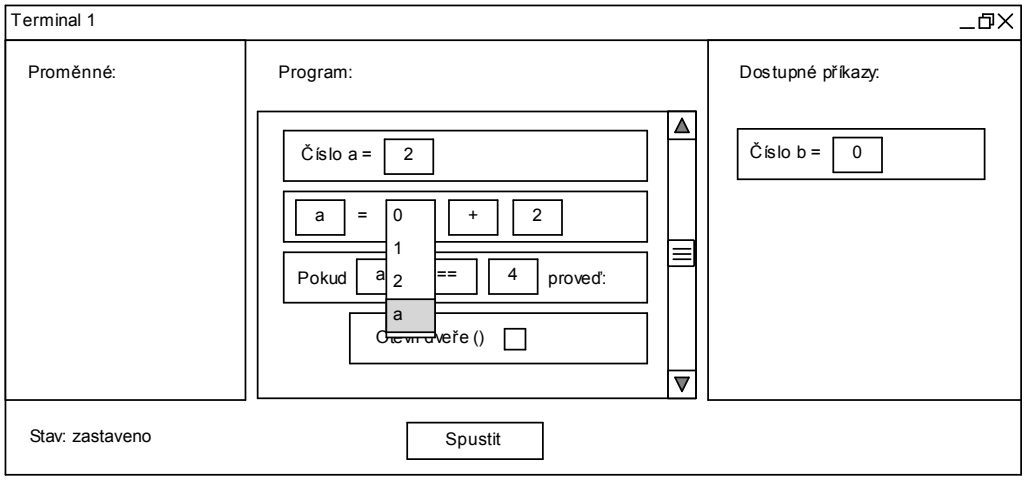
3.3.5 Jádro

Obsahuje datovou reprezentaci příkazů a instanci enginu Nashorn. Stará se o běh uživatelských programů. Je rozděleno na část uživatelského programu a dostupné příkazy. Tyto části jsou reprezentovány uspořádanými kořenovými stromy obsahujícími jednotlivé příkazy. Zahrnuje také dostupné konstanty a funkce, které jsou mezi příkazy propagovány.

Vstup a výstup uživatelských programů je reprezentován voláním metod objektů herního světa. Tato volání mohou být opatřena hráči skrytým voláním kontrolní metody, která se stará o potvrzení splnění daného cíle.

Úloha je vyřešena po úspěšném běhu programu, který splnil všechny požadované cíle. Tyto cíle jsou označeny u daných příkazů čtverečkem, který je při splnění zatržen.

Po vyřešení úlohy je do fronty v úrovni přidána scéna, pomocí které jsou následně provedeny změny v herním světě.



Obrázek 3.4: Návrh okna vývojového prostředí

3.3.6 Presenter

Je vytvořen společně s oknem. Inicializuje a spravuje jeho obsah. K inicializaci obsahu je využito parseru. Více o parsování lze nalézt v sekci 4.2.2.2 realizace. Presenter také odebírá události od jádra a okna. Na tyto události reaguje prováděním odpovídajících změn či voláním metod podle typu události.

3.3.7 Okno

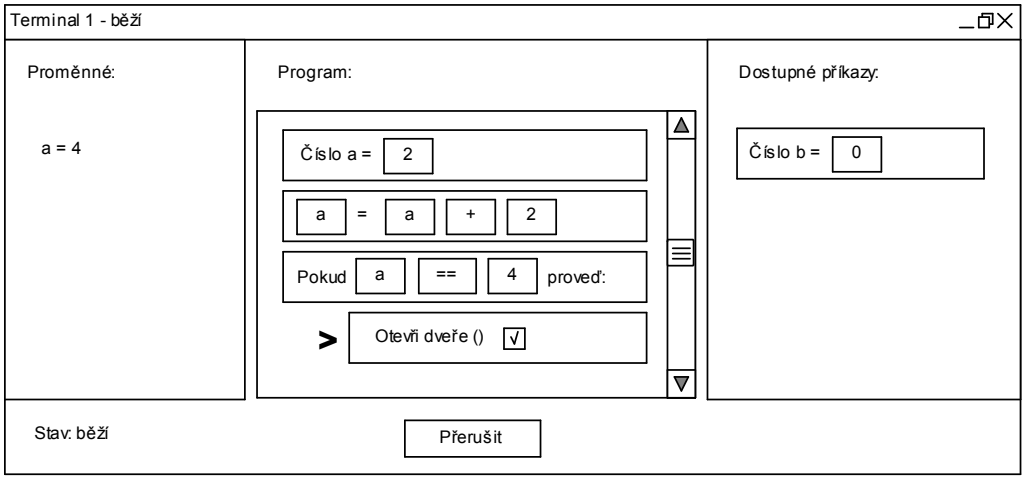
Jednotlivé příkazy jsou reprezentovány grafickými bloky. Uživatel může tyto bloky přesouvat a měnit jejich hodnoty. Výsledný program pak může zahájit za pomoci tlačítka spustit. Za běhu programu je uživateli k dispozici tlačítko přerušit, kterým může běh programu ukončit. Dále je také zvýrazněna právě prováděná část programu a zobrazena jména deklarovaných proměnných a jejich hodnoty.

Pohyb příkazů a jejich hodnoty mohou být uzamčeny. V tomto případě jsou pro uživatele označeny zámkem. Je tak učiněno ze začátku kurzu, kdy se uživatel seznamuje s prostředím a v případě, že jde o část programu, která musí být pro správné vyhodnocení splnění úlohy určitým způsobem strukturovaná či nabývat specifických hodnot.

Návrh okna je zobrazen na obrázku 3.4 a 3.5.

3.3.8 Nápovědy a popisky

Součástí vývojového prostředí jsou také informační popisky, podle kterých se hráč řídí při řešení algoritmických úloh. Dále jsou také hráči přístupné nápovědy, které jsou složeny z vysvětlivek a někdy i skryté části, kterou může odkrýt, pokud si s úlohou neví rady.



Obrázek 3.5: Návrh okna vývojového prostředí za běhu programu

3.4 Herní svět

Aplikace obsahuje dvojrozměrný svět. Tento svět se fyzicky skládá z mapy a entit. Hráč je v tomto světě reprezentován postavou, kterou může ovládat.

Herní svět je stylizován do prostředí tajné podzemní laboratoře. V této laboratoři hráč interaguje s různými interaktivními objekty. Mezi tyto objekty patří také konzole, pomocí které přistupuje do vývojového prostředí.

Návrh vzhledu herního světa je zobrazen na obrázku 3.6. Návrhový model tříd je zobrazen na obrázku 3.7.

3.4.1 Úroveň

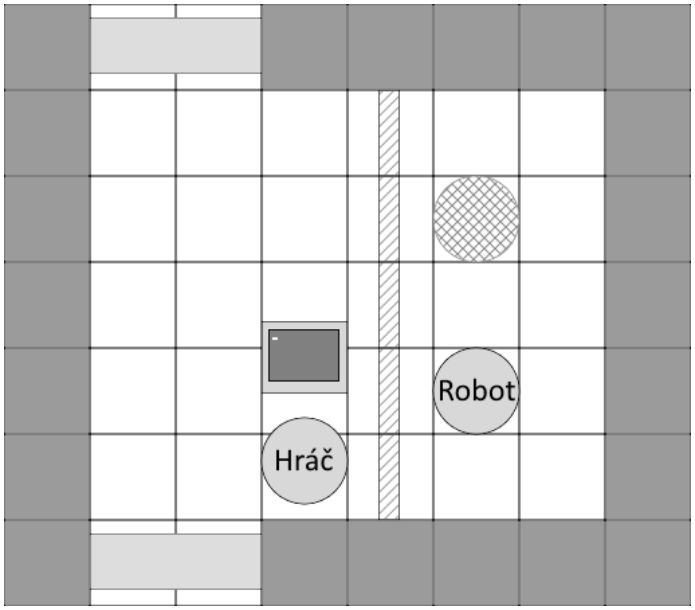
Třída `Level` je reprezentací herního světa. Obsahuje jméno, mapu, entity, vývojová prostředí a scény. Scény jsou postupně přehrávány v pořadí v jakém byly do úrovně přidány.

3.4.2 Scéna

Třída `Scene` obsahuje posloupnost akcí `IAction` uloženou ve frontě. Tyto akce zahrnují dialog `DialogAction` a interakci s objekty `InteractAction`.

3.4.3 Kolize

Součástí světa je také kolize, která se stará o to, aby hráč nemohl procházet zdmi a entitami, které jsou opatřeny kolizním objektem. Je zajištěna fyzikální knihovnou `Box2D`. Program obsahuje singleton třídy `Collision`, který se stará o správu všech kolizních objektů obsažených v úrovni a jejich interakce.



Obrázek 3.6: Návrh herního světa

3.4.4 Mapa

Je tvořena mřížkou dlaždic. Tyto dlaždice mají rozměr 64x64 pixelů. Dlaždice jsou rozděleny na podlahy a zdi. Zdi oproti podlahám zahrnují také kolizní objekt.

3.4.5 Entita

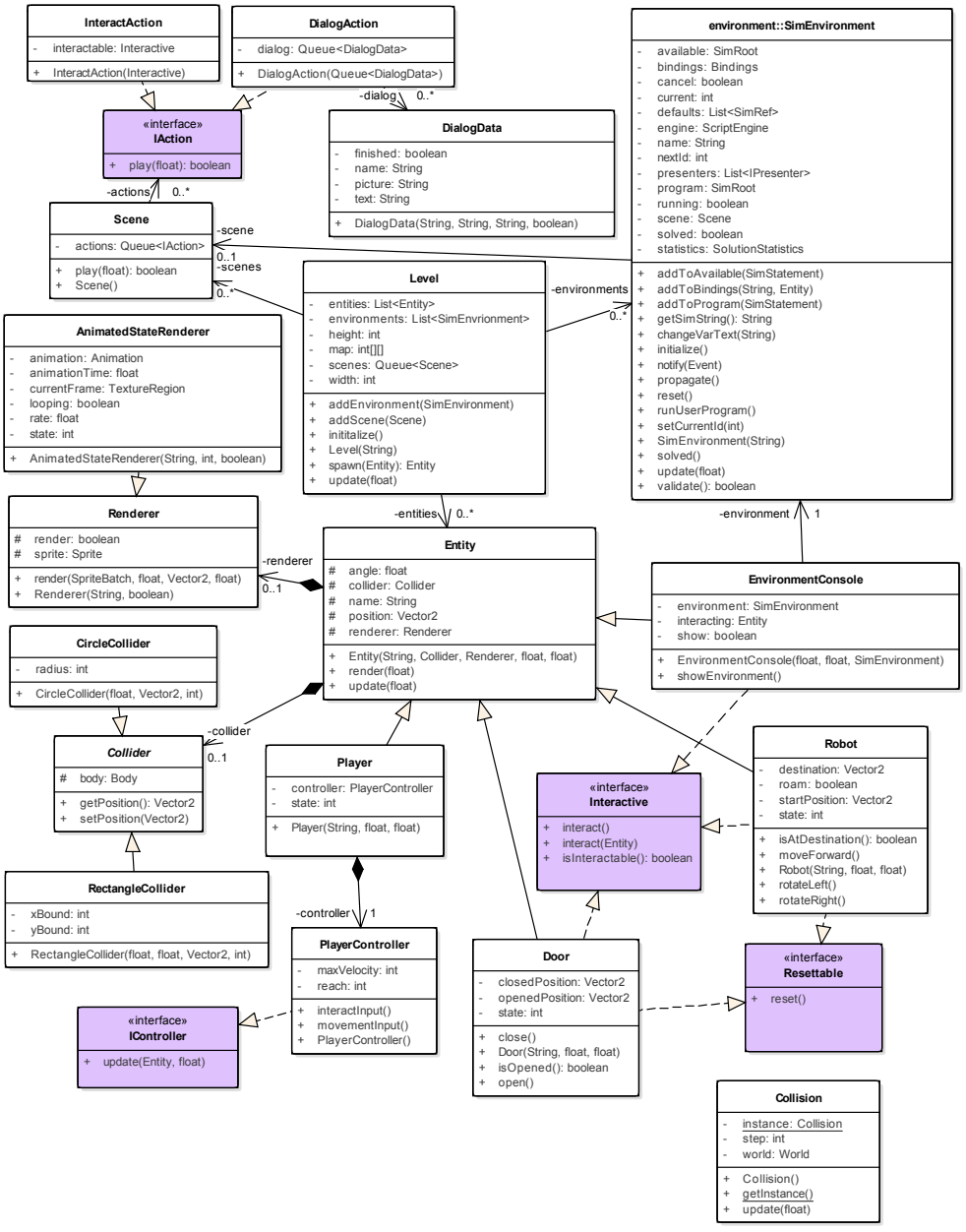
Entita je základním prvkem herního světa. Obsahuje jméno, pozici, úhel natočení, kolizní objekt a grafický objekt.

3.4.5.1 Kolizní objekt

Kolizní objekt `Collider` obsahuje fyzikální parametry entity. Může mít tvar obdélníku `RectangleCollider` nebo kruhu `CircleCollider`. Při inicializaci je přidán do kolizního systému řízeného třídou `Collision`.

3.4.5.2 Grafický objekt

Obsahuje grafickou reprezentaci entity. Může být tvořený pouze jednou texturou `Renderer` nebo animovaný se stavy `AnimatedStateRenderer`.



Obrázek 3.7: Návrhový model tříd herního světa

3.4.6 Hráčova postava

Je reprezentací studenta v herním světe. Student ji může ovládat za pomoci klávesnice a myši.

3.4.7 Robot

Může být ovládán pomocí uživatelského programu, který volá jeho metody nebo se může potulovat v dané oblasti.

3.4.8 Dveře

Pomáhají při vedení hráče kurzem. Mohou být otevřeny či zavřeny pomocí akce nebo uživatelským programem. Jejich chování je řízeno stavem.

3.4.9 Konzole

Obsahuje referenci vývojového prostředí. Pomocí interakce s konzolí si hráč zobrazuje dané vývojové prostředí.

3.5 Příběh

Součástí kurzu je také příběh, který pomáhá hráče motivovat, stanovovat cíle a objasňovat události ve hře. Tento příběh je vypravován prostřednictvím dialogu zobrazeného pomocí oken obsahujících jméno postavy, animovaný portrét a text.

3.6 Algoritmické úlohy

Žák tráví většinu času v kurzu řešením různých algoritmických úloh. S jejich pomocí také poznává různé části programování a algoritmizace. Je tedy důležité, aby byly úlohy správně navrženy, a to hlavně z hlediska obtížnosti, tak i samotné náplně.

3.6.1 Obsah

Jednotlivé úlohy byly v kurzu seřazeny tak, aby žáka postupně seznámily s jednotlivými příkazy a technikami algoritmizace. Při návrhu těchto úloh bylo přihlíženo na poznatky obsažené v [9].

Hráč se seznámí s jednotlivými částmi v následujícím pořadí:

1. proměnná
2. cyklus
3. výraz

4. podmínka
5. pole
6. funkce
7. rekurze
8. bublinkové třídění

3.6.2 Obtížnost

Hráči jsou postupně představovány složitější problémy a předávána kontrola nad celkovým obsahem programu. Žák tedy nejdříve může pouze změnit hodnotu číselné proměnné u jednoduchého programu.

Kapitola 4

Realizace

4.1 Interakce

Entity, s kterými může hráč interagovat, jsou ve hře vyhledávány za pomoci metody `rayCast` třídy `Collision`, které je předána pozice hráčovi postavy, pozice kam až může hráč dosáhnout a instance třídy `EntityRayCallback`. Kolizní objekty, kterými paprsek, vytvořený mezi danými pozicemi, prošel, jsou do `EntityRayCallback` nahrány. Pokud je nejbližší prvek kolizním objektem entity, která realizuje rozhraní `Interactive` a její metoda `isInteractable` vrací hodnotu true, je hráči nad tímto objektem zobrazena ikonka interakce. Po stisknutí klávesy `E` je zavolána metoda `interact` této entity, které je předána reference na hráčovu postavu.

4.2 Vývojového prostředí

4.2.1 Jádro

4.2.1.1 Nashorn

Hlavní částí jádra je javascriptový engine `Nashorn`. Je vybrán a inicializován za pomoci třídy `ScriptEngineManager`. K enginu je následně přistupováno přes instanci třídy `ScriptEngine`. Volá metody objektů aplikace za pomoci třídy `Bindings`, která obsahuje mapu referencí objektů a jejich identifikátory. Jedním z těchto objektů je také samotné jádro.

4.2.1.2 SimRef

Reference jsou implementovány třídou `SimRef`, která obsahuje identifikátor, typ referovaného prvku a jeho datový typ či návratový typ u funkce. Pro funkce také obsahuje počet a typ atributů, které mohou být předány.

4.2.1.3 Hodnoty příkazu

Hodnoty příkazu jsou reprezentovány třídou `SimStringInfo` pro řetězce a pro reference třídou `SimRefInfo` a jejími nástavbami. `SimRefInfo` obsahuje filtry, které předurčují jakého datového typu a typu referovaného prvku může hodnota nabývat. Pokud není dostupný žádný validní kandidát, nabývá `null`.

4.2.1.4 Funkčnost

Reference na dostupné konstanty, proměnné, funkce a pole jsou jednotlivým příkazům předávány za pomoci metody `propagate`. Tato metoda se stará o to, aby byly předány pouze reference dostupné v dané části. Je volána na kořeny obou stromů příkazů po inicializaci jádra a přesunu příkazu.

Uživatelský program složený z jednotlivých příkazů je přeložen do javascriptového kódu za pomoci zavolání metody `getSimString` na kořenový prvek uživatelského programu. Tento kód je následně interpretován za pomoci metody `eval` enginu `Nashorn`. Před spuštěním programu je vývojovému prostředí z thread poolu `ExecutorService` přiděleno vlákno, které následně volá metodu `runUserProgram` a stará se o běh daného programu.

Pro účely sledování průběhu programu uživatelem je zdrojový kód prokládán skrytými pauzami. Dále také program volá metodu `changeVarText` jádra, která pomocí vyslání události, která je odebrána presentery realizujicími rozhraní `IPresenter`, předá oknům `SimWindow` řetězec s názvy a hodnotami proměnných. Volána je také metoda `setCurrentId`, která stejným způsobem jako předchozí předá oknu id právě prováděného příkazu.

Pokud běh programu skončil neúspěšně, pak je nad všemi entitami v `Bindings` realizujícími rozhraní `Resettable` zavolána metoda `reset`, která uvede tyto entity do počátečního stavu.

4.2.2 Grafické uživatelské rozhraní

Je implementováno za pomoci knihovny `scene2d`. Hlavní částí je okno prostředí `SimWindow`, které zobrazuje jednotlivé příkazy za pomoci grafických prvků reprezentovaných třídou `SimElement` pro jednoduché příkazy a pro složené třídou `SimComplexElement`. Tyto prvky mohou obsahovat řetězce reprezentované třídou `SimLabel`, měnitelné hodnoty realizované `SimSelectBox` a obrázky `SimImage`. Dále také obsahuje informační dialogy `Dialog`, nápovědy `SimHelpTooltip` a popisky úlohy `SimTooltip`.

4.2.2.1 Přesun prvků

Jednotlivé prvky jsou přesouvány za pomoci třídy `DragAndDrop` knihovny `scene2d`. Ta v sobě uchovává cíle `Target` všech prvků, které mohou být cílem, a zdroje `Source` prvků, které nemají uzamknutý pohyb. Tento systém umožňuje uživateli myší daný prvek přenést nad validní cíl. Pokud tak hráč učiní, je

odeslána událost `ElementMoved` presenterovi, který se postará o změnu v jádře `SimEnvironment`. Jádro tuto událost oznámí všem presenterum, kteří následně aktualizují svá okna.

4.2.2.2 Parsování

Grafické prvky jsou při inicializaci okna vývojového prostředí vytvářeny za pomoci parseru, kterému jsou předány kořenové prvky `SimRoot` obou částí v jádře. Ten následně, díky uspořádané hierarchické struktuře uživatelského programu a dostupných příkazů, vytvoří odpovídající grafickou reprezentaci. Součástí vytváření je také lokalizace řetězců do právě používaného jazyka, pokud takový řetězec existuje.

Parser na základě řetězce nacházejícího se v `TextBundle` a typu daného jednoduchého příkazu sestaví obsah odpovídajícího grafického prvku realizovaného třídou `SimElement` z posloupnosti řetězců `SimLabel`, obrázků `SimImage`, rozbalovacích nabídek `SimSelectBox` a jejich nástaveb. Řetězec, obsažený v `TextBundle`, může zahrnovat mimo samotného textu také reference na různé parametry příkazu složené ze znaku `$` a názvu daného parametru. Potřebná data tohoto parametru jsou odebrána od příkazu za pomoci metody `getParameter`. Na základě typu předaných dat je zvolena příslušná grafická reprezentace. Pokud jde o příkaz složený, je vytvořena instance třídy `SimComplexElement` a její obsah sestaven z jednoho nebo více `SimElement`.

Implementace tohoto systému usnadnila práci na samotném vývojovém prostředí, protože odpadla potřeba vytvoření tříd reprezentujících specifické grafické prvky pro každý příkaz. Její výhodou je také možnost vytvoření více variant těchto prvků pro jeden příkaz a možnost přeskládání parametrů příkazu podle použitého jazyka.

4.2.3 Kontrola chyb

Kontrola chyb je zajištěna za pomoci různých podmínek, které se kontrolují při pokusu o spuštění uživatelského programu a při běhu programu.

Jakmile chce uživatel svůj sestavený program spustit, tak je rekurzivně zavolána metoda `validate` na všechny prvky uživatelského programu. Pokud nějaký prvek obsahuje nepovolenou hodnotu `null`, pak je místo spuštění programu zobrazen informační dialog realizovaný třídou `Dialog`.

Při běhu program volá uživateli skryté funkce, které kontrolují možné logické chyby. Tyto funkce zahrnují:

- kontrolu přístupu do prvku mimo pole

  Před přístupem do prvku pole je zavolána funkce `getField`, která zkontroluje jestli předaný index zasahuje mimo dané pole. Pokud k tomu dojde, je program přerušen a zobrazen informační dialog.

- kontrolu přístupu do prvku mimo řetězec

  Při využití funkce `strCharAt` je zkontrolováno jestli předaný index zasahuje mimo daný řetězec. Pokud zasahuje, je program přerušen a zobrazen informační dialog.

- kontrolu nekonečného cyklu

  Všechny části, které mohou být zacykleny, jsou opatřeny kontrolní proměnnou, která se při každém zavolání zvýší o jedna. Pokud tato proměnná dosáhne specificky nastaveného limitu je program přerušen a zobrazen informační dialog.

- kontrolu dělení nulou

  Před provedením dělení je zavolána funkce `testForZero`, která zkontroluje hodnotu dělitele. Pokud se rovná nule je program přerušen a zobrazeno upozornění.

Všechny dialogy, které jsou touto kontrolou uživateli zobrazeny, jsou obsaženy v obrázku 4.1.

4.3 Persistence dat

Systém persistence dat v třídě `FileIO` je implementován za pomoci technologie zápisu dat a knihovny `JSON`. Třída obsahuje metody a předpisy pro serializaci a následné nahrání určitých dat některých složitějších objektů. Zároveň také obsahuje předpisy pro opětovnou inicializaci nahraných objektů. Aplikace ukládá své soubory do složky `.a16o` v domovské složce přihlášeného uživatele. Těmito soubory jsou:

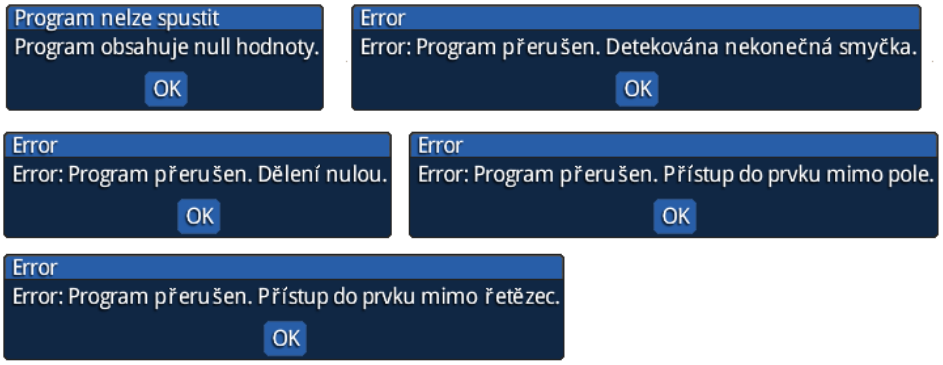
- Profil hráče

  Je označen názvem `profile.sav` a jde o uložená data instance třídy `Profile`, která obsahuje jméno hráče, hodnoty nastavení a jednotlivé statistiky průchodu.

- Pozice hráče v kurzu

  Je označena názvem `progress.sav` a obsahuje data instance třídy `Level`, která jsou potřebná pro znovuobnovení činnosti po nahrání z tohoto souboru.

Jelikož zápis `JSON` je přímo čitelný, jsou všechny soubory vytvořené aplikací kódovány za pomoci `Base64`, které je přímo obsažené ve frameworku `LibGDX`. Je tak učiněno z důvodu nasazení aplikace ve školním prostředí, kde lze předpokládat možné snahy o alteraci uložených dat. Jelikož je prioritou především nečitelnost uložených údajů, a ne celkové zabezpečení dat, není využito šifrování.



Obrázek 4.1: Dialogy kontroly chyb

4.4 Lokalizace

Aplikace využívá systému lokalizace všech obsažených textový řetězců třídou `I18NBundle`, která je součástí frameworku `libGDX`. Lokalizace zahrnuje překlad do českého a anglického jazyka. Textové řetězce jsou pro každý jazyk uloženy ve tvaru identifikátor = řetězec v souboru `TextBundle`<_označení jazyka>`.properties` ve vnitřní složce `assets` a jsou přístupné za pomoci předání identifikátoru metodě `get`. Anglický jazyk je nastaven třídou `I18NBundle` jako výchozí jazyk a jeho řetězce jsou v souboru `TextBundle.properties`. Český jazyk je uložen v souboru `TextBundle_cs_CZ.properties`.

4.5 Zvuk

Aplikace neobsahuje zvukovou stopu z důvodů uvedených v části 2.1.1 analýzy.

4.6 Technologie a nástroje

Při tvorbě různých částí didaktické hry bylo využito množství nástrojů a technologií. Do této kategorie patří jak nástroje pro tvorbu grafických zdrojů, návrh a vývoj samotné aplikace, tak i operační systém, pod kterým byl program vytvořen i testován. Je důležité uvést, že až na operační systém Windows a program Enterprise Architect, jsou použité nástroje dostupné zdarma ke stažení na internetu a některé jsou také open-source.

4.6.1 Windows [10]

Aplikace byla vyvíjena a testována pod operačním systémem Windows 8.1.

4.6.2 Java [11]

Java je objektově orientovaný programovací jazyk, který umožňuje efektivní multiplatformní vývoj aplikací díky použití virtuálního stroje pro běh vytvořených programů. Při vývoji aplikace bylo využito verze 8u66.

4.6.3 JavaScript [12]

JavaScript je objektově orientovaný skriptovací jazyk. Byl využit při realizaci uživatelských programů enginem Nashorn.

4.6.4 Eclipse [13]

Eclipse je open source vývojová platforma. Při vývoji a testování této práce bylo využito vývojového prostředí verze Mars.1 (4.5.1) určené pro programování v jazyce Java.

4.6.5 LibGDX [14]

LibGDX je framework pro vývoj multiplatformních aplikací v Javě. Tento framework obsahuje množství užitečných knihoven a nástrojů.

4.6.6 Nashorn [15]

Nashorn je javascriptový engine implementovaný v Javě, který je součástí distribuce Javy od verze 8. Jeho užitečnou vlastností je schopnost interpretace kódu za běhu aplikace.

4.6.7 JSON [16]

JSON je javascriptový objektový zápis, který je nezávislý na počítačové platformě a umožňuje jednoduchý a zároveň efektivní způsob zápisu dat.

4.6.8 Paint.NET [17]

Paint.NET je grafický editor obrázků a fotografií. Program obsahuje množství užitečných editačních nástrojů, efektů, štětců a palet. Výhodou tohoto programu je intuitivnost a jednoduchost ovládání.

4.6.9 Enterprise Architect [18]

Enterprise Architect je komplexním grafickým UML nástrojem užitečným pro výrobu všech potřebných částí dokumentace celého životního cyklu vývoje systému.

4.7 Dokumentace

Důležitou součástí vývoje je udržení srozumitelnosti a čitelnosti zdrojového kódu pro pozdější možnost úprav či využití, a to jak autorem, tak i vývojářem, který daný zdrojový kód vidí zcela poprvé. Srozumitelnosti napomáhají správně zvolené názvy metod a tříd a okomentování důležitých či složitých metod a tříd. Z tohoto důvodu práce obsahuje dokumentaci vygenerovanou z okomentovaného zdrojového kódu za pomoci nástroje `Javadoc`. Tuto dokumentaci lze nalézt na přiloženém médiu.

4.8 Testování

Nedílnou součástí vývoje aplikace je také testování její funkčnosti na možné chyby vzniklé při programování. Testování je rozděleno do dvou kategorií.

4.8.1 Manuální testování

Aplikace byla v průběhu vývoje vlastnoručně testována autorem. Zmíněné testování sestávalo z manuálního otestování jednotlivých komponent programu. Všechny nalezené chyby byly následně zaevidovány a opraveny.

4.8.2 Automatické testování

Samotný program také obsahuje unit testy důležitých komponent. Unit testy jsou krátké kusy kódu, kterými je verifikována funkčnost a korektnost dané komponenty. Všechny nalezené chyby byly opraveny.

Závěr

Cílem práce bylo analyzovat, navrhnout a implementovat interaktivní výukový program zaměřený na téma algoritmizace úloh. Tento program měl umožňovat uživateli sestavovat, spouštět a sledovat průběh programů, které řeší algoritmické úlohy obsažené v kurzu. Součástí měla být také evidence statistik o hráčově postupu a herní svět, v kterém se celý kurz odehrává. V rámci bakalářské práce byly tyto cíle splněny.

Aplikace je v současném stavu použitelná jakožto pomůcka pro výuku, která uživatele uvede do algoritmizace a programování. Zahrnuje kurz skládající se z dvaceti algoritmických úloh. Pokud by aplikace měla být využita jako hlavní či jediný prvek výuky, musela by být rozšířena o množství přídavných prvků.

Osobní přínos

Prací na projektu autor nabyl nové užitečné zkušenosti s vývojem herních aplikací jako takových, tak i aplikací zaměřených na výuku. Těchto zkušeností autor jistě využije v budoucí činnosti.

Výhled do budoucna

Autor by rád v budoucnu na projektu dále pracoval. Projekt je možné rozšířit o vlastní editor úrovní, který by umožnil vytváření nových kurzů. Součástí tohoto rozšíření by bylo i provedení potřebných změn v aplikaci. Další možností je přidání nové funkčnosti a příkazů do vývojového prostředí. Překlad do nového jazyka je také možností.

Použité zdroje

[1] *Rámcový vzdělávací program pro gymnázia*. Praha: Výzkumný ústav pedagogický v Praze, 2007, ISBN 978-80-87000-11-3, 65 s., [cit. 2015-11-04]. Dostupné z: http://www.vuppraha.cz/wp-content/uploads/2009/12/RVPG-2007-07_final.pdf

[2] DOSTÁL, J.: Výukový software a počítačové hry - nástroje moderního vzdělávání. *JTIE*, ročník 1, č. 1, 2009: s. 23–28, ISSN 1803-537X, [cit. 2015-11-15]. Dostupné z: http://jtie.upol.cz/cz/artkey/jti-200901-0003.php

[3] DVORSKÝ, J.: *Algoritmy I.* Ostrava: VŠB, 2007, [cit. 2015-11-18]. Dostupné z: http://www.cs.vsb.cz/dvorsky/Download/SkriptaAlgoritmy/Algoritmy.pdf

[4] KNUTH, D.: *Umění programování*. Brno: Computer press, první vydání, 2008, ISBN 978-80-251-2025-5.

[5] VIRIUS, M.: *Základy algoritmizace*. Praha: ČVUT, první vydání, 1995, ISBN 80-01-01346-4, [cit. 2015-11-20]. Dostupné z: www.rudisweb.wz.cz/dokumenty/algoritmizace.pdf

[6] MALONEY, J. H.; Peppler, K.; Kafai, Y.; aj.: Programming by Choice: Urban Youth Learning Programming with Scratch. *SIGCSE Bull.*, ročník 40, č. 1, Březen 2008: s. 367–371, ISSN 0097-8418, [cit. 2015-11-27]. Dostupné z: https://www.cs.swarthmore.edu/~turnbull/cs91/f09/paper/maloney08.pdf

[7] DANN, W.; Cooper, S.; Pausch, R.: Making the Connection: Programming with Animated Small World. *SIGCSE Bull.*, ročník 32, č. 3, Červenec 2000: s. 41–44, ISSN 0097-8418, [cit. 2015-11-22]. Dostupné z: http://web.stanford.edu/~coopers/alice/iticse00.pdf

[8] FOWLER, M.: *GUI Architectures*. [online], July 2006, [cit. 2015-11-29]. Dostupné z: `http://martinfowler.com/eaaDev/uiArchs.html`

[9] HANSEN, P. B.: *Programming for everyone in Java*. New York: Springer-Verlag, 1999, ISBN 978-1-4612-1514-1.

[10] MICROSOFT CORPORATION: *Windows*. [software], [cit. 2015-12-02]. Dostupné z: `https://www.microsoft.com/cs-cz/windows`

[11] ORACLE CORPORATION: *Java*. [software], [cit. 2015-12-02]. Dostupné z: `https://www.java.com`

[12] NETSCAPE COMMUNICATIONS: *JavaScript*. [online], [cit. 2015-12-02]. Dostupné z: `https://developer.mozilla.org/cs/docs/Web/JavaScript`

[13] ECLIPSE FOUNDATION: *Eclipse*. [software], [cit. 2015-12-02]. Dostupné z: `https://eclipse.org/`

[14] BADLOGIC GAMES: *libGDX*. [software], [cit. 2015-12-02]. Dostupné z: `https://libgdx.badlogicgames.com/`

[15] ORACLE CORPORATION: *Nashorn*. [online], [cit. 2015-12-02]. Dostupné z: `http://openjdk.java.net/projects/nashorn/`

[16] CROCKFORD, D.: *JSON*. [online], [cit. 2015-12-02]. Dostupné z: `http://json.org/`

[17] DOTPDN LLC: *Paint.NET*. [software], [cit. 2015-12-02]. Dostupné z: `http://www.getpaint.net/index.html`

[18] SPARX SYSTEMS: *Enterprise Architect*. [software], [cit. 2015-12-02]. Dostupné z: `http://www.sparxsystems.com.au/products/ea/index.html`

Příloha **A**

Seznam použitých zkratek

RVP-G rámcový vzdělávací program

LIFO last in first out

FIFO first in first out

UML unified modeling language

2D two-dimensional

3D three-dimensional

JSON javascript object notation

Příloha **B**

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `a16o.jar` ..... spustitelná aplikace
- `javadoc` ..... dokumentace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF

Příloha C

Uživatelská příručka

C.1 O aplikaci

A16o je didaktickou počítačovou hrou zaměřenou na téma algoritmizace úloh. V rámci této hry se uživatel účastní interaktivního kurzu, při kterém řeší různé algoritmické úlohy za pomoci programů, které sestavuje a spouští.

C.2 Profil hráče

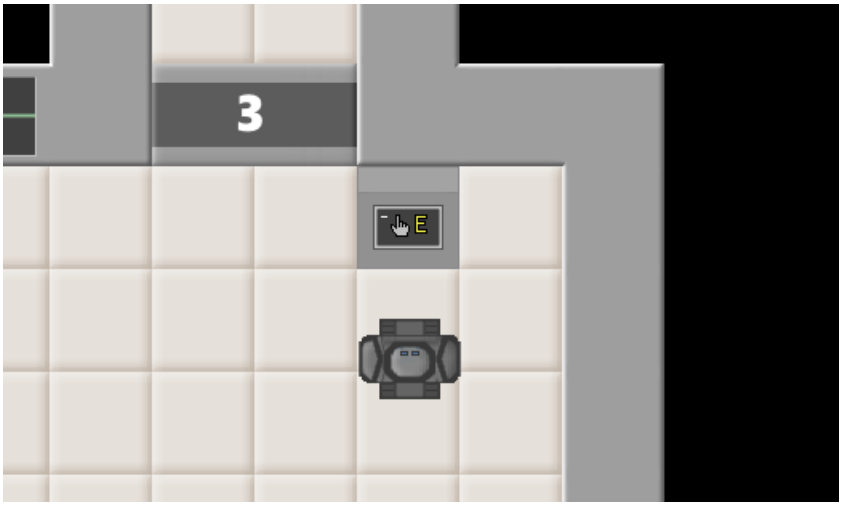
Při prvním spuštění aplikace si hráč vytvoří profil zadáním jména splňujícího podmínku délky alespoň dvou znaků a stisknutím tlačítka Vytvořit. Obsah svého profilu, který zahrnuje jméno a statistiky hráče, může hráč zobrazit stisknutím tlačítka Profil hráče v hlavní nabídce.

C.3 Změna jazyka

Aplikace obsahuje lokalizaci do anglického a českého jazyka. Pro změnu jazyka je třeba v hlavní nabídce hry stisknout tlačítko Nastavení a následně zmáčknout na tlačítko změny jazyka.

C.4 Ovládání ve hře

Za běhu je možné hru uložit stiskem klávesy `F5` a nahrát stiskem klávesy `F6`. Dále je možné vrátit se do hlavní nabídky stiskem klávesy `ESC`. Tyto možnosti jsou hráči přístupné pouze, pokud v daném okamžiku neběží žádný uživatelský program.



Obrázek C.1: Ikona interakce

C.5 Ovládání postavy

C.5.1 Otáčení

Postavou může hráč otáčet za pomoci pohybu kurzoru myši po obrazovce, který postava sleduje.

C.5.2 Pohyb

Postavou může hráč pohybovat za pomoci kláves `W S A D`.

C.5.3 Interakce

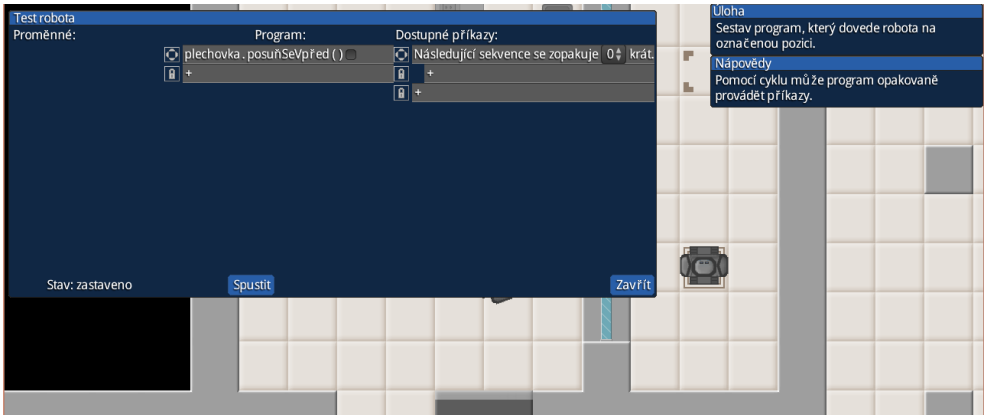
Jestliže je hráčova postava namířena směrem k objektu, s kterým lze interagovat, a je v dostatečné blízkosti, pak je zobrazena ikonka interakce viz C.1. V takovém případě může hráč stisknutím klávesy `E` s tímto objektem interagovat.

C.6 Vývojové prostředí

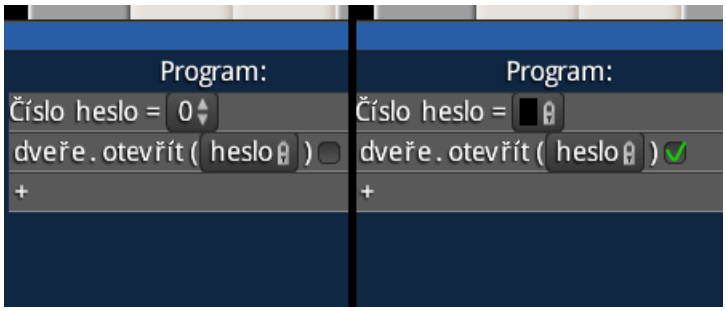
Vývojového prostředí uživatel využívá k sestavení a spuštění programů řešících různé úlohy. Vývojové prostředí lze vidět na obrázku C.2.

C.6.1 Spuštění programu

Zhotovený program je možné spustit stiskem tlačítka Spustit.



Obrázek C.2: Vývojové prostředí



Obrázek C.3: Požadovaný cíl před a po úspěšném běhu

C.6.2 Přerušení programu

Běžící program je možné zastavit stiskem tlačítka Přerušit.

C.6.3 Zavření okna

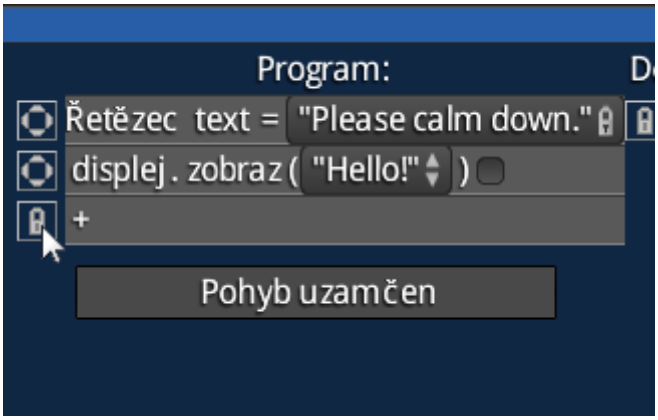
Okno vývojového prostředí lze zavřít za pomoci tlačítka Zavřít.

C.6.4 Požadované cíle

Programy vytvářené ve vývojovém prostředí musí pro vyřešení dané úlohy splnit zadání a požadované cíle. Splnění těchto cílů lze sledovat za běhu programu za pomoci ikonek u důležitých příkazů viz obrázek C.3. Jestliže jsou na konci běhu programu všechny ikonky zatrženy, je úloha splněna.

C.6.5 Stav programu

Stav programu je zobrazen v levém dolním rohu okna. Program může nabývat stavu zastaveno, kdy je možné přesouvat a měnit hodnoty programu. Dále



Obrázek C.4: Zamknutý příkaz

může nabývat stavu běží, kdy je daný program spuštěný a uživatel může sledovat jeho průběh, a stavu vyřešeno, kdy byl požadovaný program sestaven, spuštěn a program za svůj běh splnil všechny požadované cíle.

C.6.6 Přesun příkazu

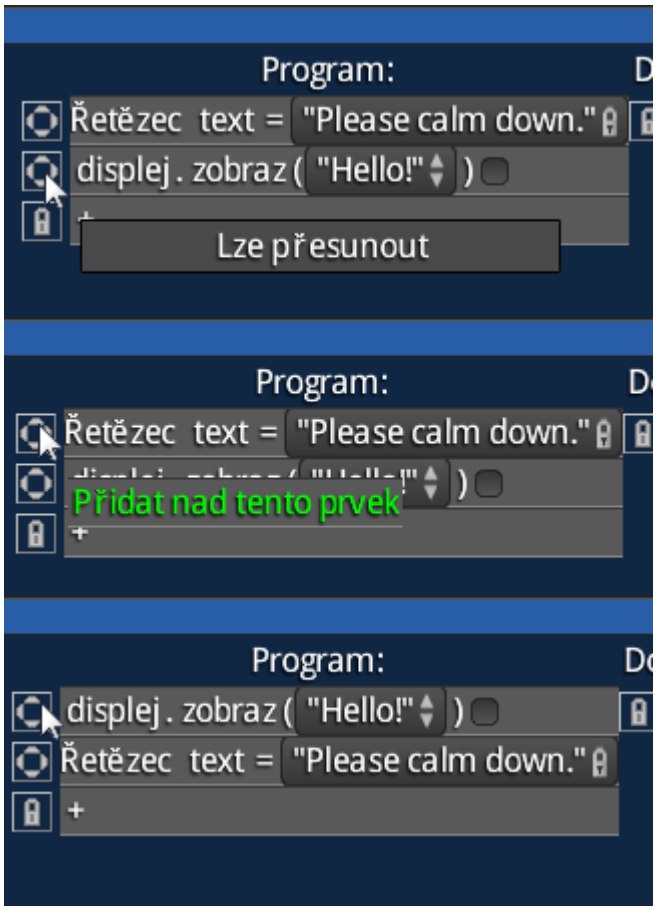
Příkaz je možné přesunout, pokud není jeho pohyb zamknutý, čehož si lze všimnout za pomoci ikonky vlevo od něj viz obrázek C.4. Samotný přesun je realizován stisknutím a podržením levého tlačítka myši s kurzorem nad daným příkazem a následným přesunutím kurzoru nad validní prvek a puštěním levého tlačítka myši. Pokud jde o validní prvek, je zobrazen zelený text „Přidat nad tento prvek“ viz C.5.

C.6.7 Změna hodnoty v příkazu

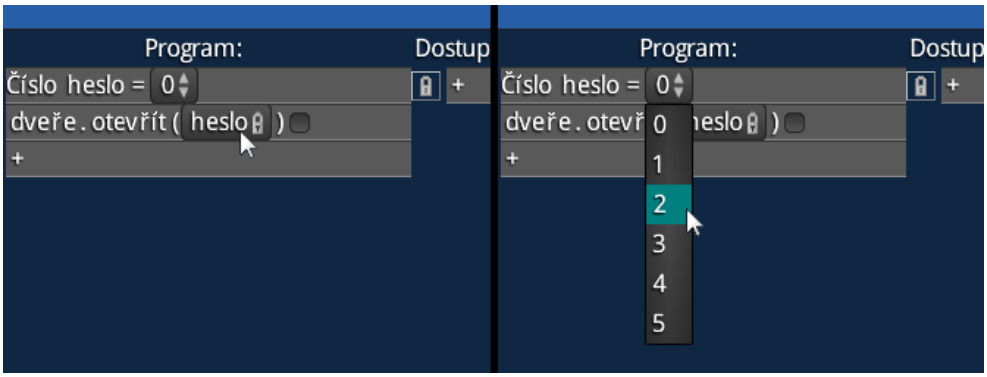
Hodnotu v příkazu lze změnit, jestliže tato hodnota není zamknuta viz obrázek C.6. Kliknutím levého tlačítka myši s kurzorem nad touto hodnotou se rozbalí nabídka. Novou hodnotu lze vybrat za pomoci stisknutí levého tlačítka myši s kurzorem nad hodnotou v nabídce.

C.6.8 Nápověda

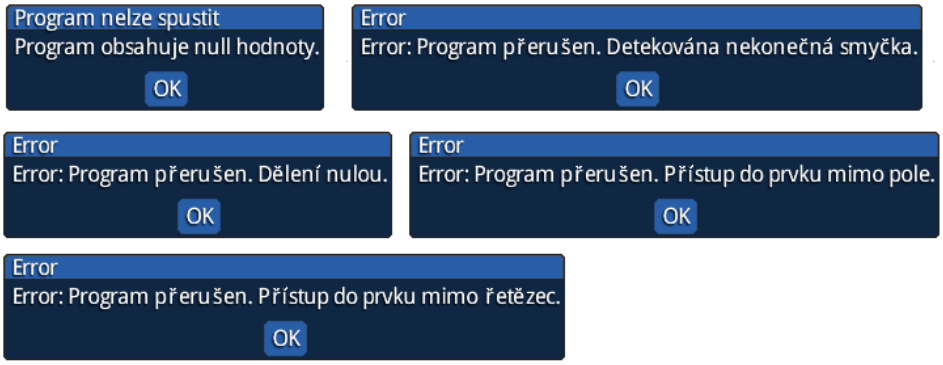
V rámci řešení úloh může hráč využít nápovědy, pokud je v dané úloze dostupná. Této nápovědy by měl využít jen v krajní nouzi, jelikož je její využití zaznamenáno do jeho profilu. Nápovědu je možné zobrazit stisknutím tlačítka Odkrýt v okně nápovědy.



Obrázek C.5: Přesun příkazu



Obrázek C.6: Zamknutá hodnota a vybrání hodnoty



Obrázek C.7: Dialogy kontroly chyb

C.6.9 Informační dialogy

Jestliže vytvořený program obsahuje chyby při spuštění, či jsou chyby detekovány za běhu, je uživatel upozorněn za pomoci informačních dialogů. Tyto dialogy jsou vyobrazeny na obrázku C.7.