Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Študijný program: Aplikovaná informatika

Obor: Informatika

Počítačová hra pre vstupné kurzy programovania v jazyku Java

BAKALÁRSKA práca

Študent : Dávid Bobula

Vedúci : Ing. Rudolf Pecinovský, CSc

Praha

máj 2019

Prehlásenie:

Prohlašuji, že jsem bakalářskou práci Počítačová hra pro vstupní kurzy programování v jazyku Java zpracoval za použití v práci uvedených pramenů a literatury.

V Praze dne 06. 05. 2019 . . . . . .

Dávid Bobula

Poďakovanie

Chcel by som sa poďakovať pánovi profesorovi Ing. Pecinovskému za čas, ochotu a trpezlivosť, ktorú mi venoval pri tvorbe tejto bakalárskej práce.

Abstrakt

Cílem této bakalářské práce je vytvoření návrhu hry, kterou by bylo možné použít při výuce ve vstupním kurzu programování, a to konkrétně na Vysoké škole ekonomické v Praze na studijním oboru Aplikovaná informatika na fakultě Informatiky a Statistiky.

V práci jsou analyzovány přístupy k postupně vylepšovaným programům v průběhu výuky programování na různých školách. Práce též rozebírá vhodné jazyky a vývojová prostředí pro výuku programování. Na základě této analýzy je následně připravený návrh vlastní hry s využitím metody postupně vylepšovaného programu.

Klíčová slova

Hry, Java, výuka programování.

Abstrakt

Cieľom tejto bakalárskej práce je vytvorenie návrhu hry, ktorú by bolo možné použiť pri výuke vo vstupnom kurze programovania, a to konkrétne na Vysokej škole ekonomickej v Prahe v študijnom obore Aplikovaná informatika na fakulte Informatiky a Štatistiky.

V práci sú analyzované prístupy k postupne vylepšovaným programom v priebehu výuky programovania na rôznych školách. Práca tiež rozoberá vhodné jazyky a vývojové prostredia pre výuku programovania. Na základe tejto analýzy je následne pripravený návrh vlastnej hry s využitím metódy postupne vylepšovaného programu.

Kľúčové slová

Hry, Java, výuka programovania.

Abstract

The goal of this bachelor thesis is to create design of game, which could be used in teaching in beginner course of programming, specifically at University of economics in Prague, in learning course Applied Informatics at Faculty of Informatics and Statistics.

In the thesis are analyzed approaches to continuously improving programs during learning of programming at different schools or universities. The thesis also consists of summary of proper languages and development environments for learning of programming. Based on this analysis, there is subsequently created design for own game with usage of method of continuously improving program.

Keywords

Games, Java, programming learning.

Obsah

Úvod

Tému práce som si zvolil z navrhovaných tém bakalárskych prác, z dôvodu záujmu o programovanie. Zámerom je vytvorenie hry, ktorá by bola využitá ako výukový materiál vstupného kurzu programovania, na fakulte Informatiky a Štatistiky na Vysokej škole Ekonomickej v Prahe.

Prácu na vlastnom projekte, ktorý sa počas celej doby výuky nemení považujem za dobrú prípravu pre študentov. Študenti pracujú len na jednom projekte, s ktorým sa teda dokážu oboznámiť lepšie, na rozdiel od rôznych príkladov zameraných na jedno cvičenie a jednu tému, s ktorými sa už neskôr nestretnú.

V práci najskôr v krátkosti uvádzam súčasný stav výuky programovania a problémy, ktoré sa počas tejto výuky môžu vyskytovať. Následne rozoberám konkrétne vyučovacie metódy programovania na rôznych školách.

Ďalej analyzujem rôzne programovacie jazyky a vývojové prostredia, ktoré sa najčastejšie využívajú, nie len počas výuky, ale používané sú aj v praxi širokou verejnosťou. Keďže v praktickej časti tvorím návrh hry ako pomôcku pri výuke, uvádzam aj stručný úvod do programovania hier. Účelom je znázornenie toho, že aplikácia, ktorú študenti tvoria počas výuky, nie je len pre nutnosť udržať ich záujem, ale niečo využiteľné aj v praxi. Na záver teoretickej časti uvádzam zvažované návrhy, ktoré by sa dali použiť v priebehu výuky programovania.

Praktická časť práce teda samotná hra bola vyvíjaná v prostredí Eclipse. Eclipse som zvolil kvôli osobnej skúsenosti, no použiť sa môže aj iné prostredie ako napríklad NetBeans.

Ciele práce

- Analyzujte námety, ktoré sa vo vstupných kurzoch programovania používajú ako základ postupne vylepšovaných programov.
- Navrhnite vlastný námet pre takýto výukový program.
- Navrhnite postup výuky pri postupnom rozpracovávaní vyššie navrhnutého námetu.
- Pripravte postupnosť postupne zdokonaľovaných projektov použitých v jednotlivých krokoch tohoto postupu. Pre generovanie projektu využite program Monolith2Projects.

Cieľová skupina

Cieľovou skupinou tejto práce sú školitelia, vyučujúci vstupné kurzy programovania na školách. V práci sú pre nich poskytnuté informácie o rôznych spôsoboch výuky na iných školách, ale aj vhodné jazyky či editory pre výuku.

1 Komentovaná rešerše informačných zdrojov

1.1 Knižné zdroje

CLAYBERG, Eric, RUBEL, Dan. Eclipse plug-ins (3rd Edition). Boston: Pearson Education, Inc., 2009. ISBN 0321553462.

Anotácia:

Eclipse je viac ako popredné integrované vývojové prostredie pre prácu s Javou, je to platforma s otvoreným kódom pre tvorbu rôznych klientských aplikácií. Používaním plug-inov, môže každý vytvoriť vlastné nástroje, ktoré sú prepojiteľné s prostredím Eclipse. Táto kniha uvádza a osvetľuje celý proces vývoju plug-inov, prezentuje najlepšie hodnotené skúsenosti potrebné na dosiahnutie vysoko kvalitných výsledkov.

PECINOVSKÝ, Rudolf. OOP - Learn Object Oriented Thinking & Programming, Řepín-Živonín, Tomáš Bruckner, 2013, ISBN 978-80-904661-9-7

Anotácia:

Kniha písaná štýlom dialógu medzi záujemcom o znalosť programovania a autorom. Kniha určená úplným začiatočníkom, ktorým autor neposkytuje len teoretické poznatky jazyku Java ale učí ich skutočnému programovaniu, ako myslieť a navrhovať ako profesionálny programátor.

1.2 Odborné práce

NOVÁK, Marek, BIŇAS, Miroslav, MICHALKO, Miroslav, JAKAB, František. Ako motivovať študentov softverového inžinierstva, Technická univerzita Košice

Anotácia:

Práca pojednávajúca o spôsobe výuky programovania na vysokých školách. Informuje o možnom podceňovaní pri tejto výuke, kde sa vyučujúci sústreďuje na menej dôležité témy. Poukazuje na metodiky, ktoré výučbu programovania dokážu zjednodušiť a študenti si vyučované princípy dokážu lepšie osvojiť a porozumieť im.

PECINOVSKÝ, Rudolf. Výuka programování pro praxi, Vysoká škola ekonomická v Praze, Fakulta informačních technologií

Anotácia:

Príspevok poukazujúci na rozpory medzi vyučovaným materiálom v kurzoch programovania a očakávanými znalosťami, ktoré si vyžaduje prax.

FERGUSON, Andrew. A History of Computer Programming Languages

Anotácia:

Stručný report o vývoji programovacích jazykov, ich stručná charakteristika a ich využitie. Pojednáva o programovacích jazykoch ako FORTAN, Algol, Pascal, C, C++, Visual Basic, Python či Java.

2 Výuka programovania

Táto časť je určená ako uvedenie do témy výuky programovania vo vstupných kurzoch programovania na školách. V tejto časti je uvedený súčasný stav výuky programovania a problémy, ktoré sa s výukou spájajú.

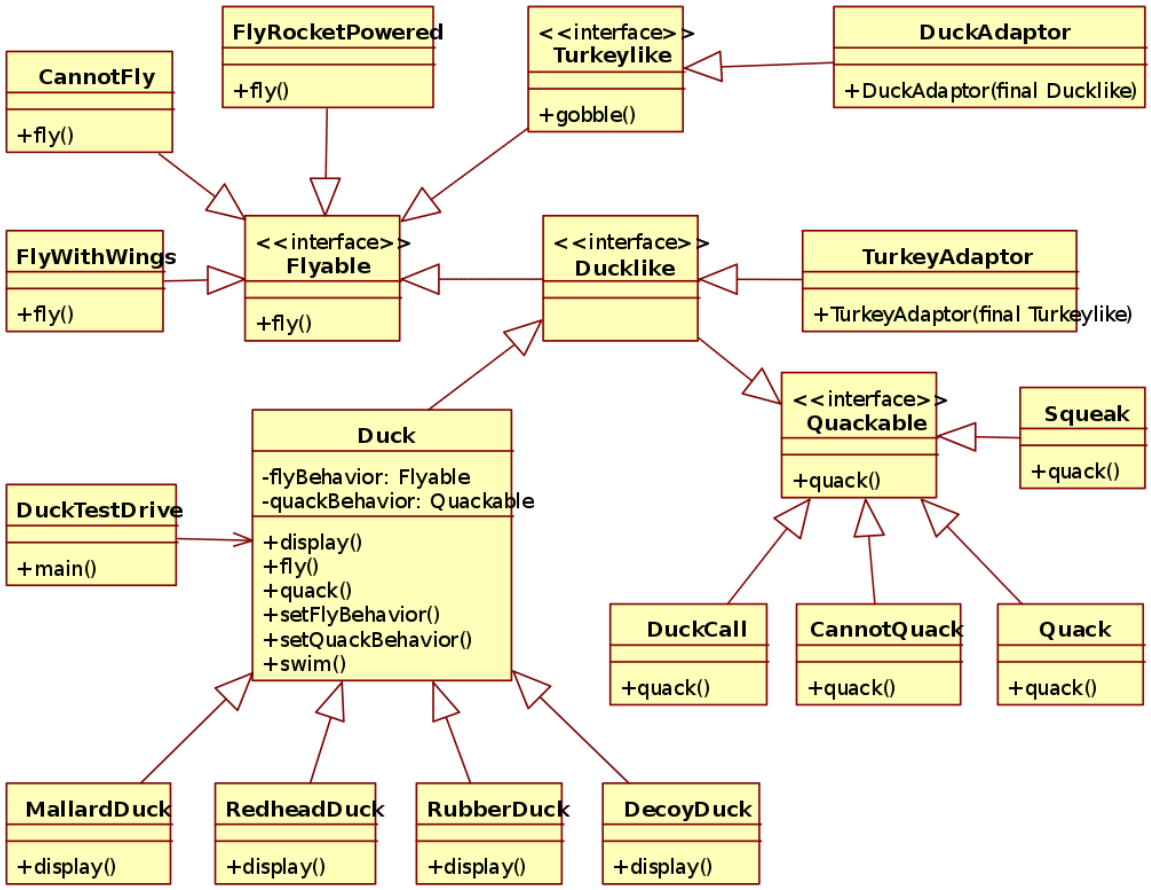
2.1 Súčasný stav

V súčasnosti sa pri vývoji softvéru v najväčšej miere používa objektovo orientované programovanie, umožňujúce vytváranie moderných aplikácií, ktoré spĺňajú nároky používateľov a taktiež umožňujú relatívne jednoduchú možnosť rozšírenia funkcionality pomocou použitia už existujúcich objektov. Pri výučbe je však potrebné nahliadať na objektovo orientované programovanie ako na paradigmu programovania a nie ako na konkrétny jazyk, ktorý používame pri výučbe. Už v začiatku kurzu je potrebné definovať študentom čo od výučby základov programovania očakávať. Študenti ktorý navštevujú kurz objektovo orientovaného programovania často nadobudnú pocit, že podobný kurz ich naučí vytvárať plnohodnotné aplikácie s vlastným grafickým používateľským prostredím, čo nie je cieľom takýchto kurzov. Ich cieľom by malo byť oboznámiť študentov so základnými vlastnosťami a princípmi objektového programovania a neskôr s použitím zvoleného programovacieho jazyku demonštrovať tieto možnosti v praxi. Taktiež je vhodné oboznámiť začínajúcich študentov s vývojovými prostrediami, ktoré im umožnia lepšie pracovať s daným jazykom a jeho kódom a taktiež umožnia jednoduchšie odhalenie chýb v kóde.[1]

2.2 Problémy spojené s výukou

Pri výučbe objektovo orientovaného programovania samozrejme narážame na rôzne problémy, ktoré sú spojené s nesprávnym prístupom vyučujúcich k danej problematike. Školitelia často používajú nevhodné príklady, ktoré študentom často dávajú mylnú predstavu o technológii, ktorú sa snažia naučiť. Použitie podobného obrázku ako na *obrázku 1* pri opisovaní objektovo orientovaného programovania je nešťastné, nakoľko študent, ktorý len teraz prichádza do styku s podobnou technológiou nemá dostatočný

rozhľad na to aby pochopil čo sa mu školiteľ snaží vysvetliť.[1]



Obrázok 1: Diagram znázorňujúci dizajn správania zvieraťa podľa schopnosti lietať a vydávať zvuk

Pri precvičovaní nadobudnutých znalostí nastáva opačný problém, keď používame príliš zjednodušené scenáre, ktoré sa približujú primitívnym príkladom typu „Hello world!“ čím si síce študenti precvičia syntax daného jazyka avšak podobné príklady nepredstavujú výzvu pre študentov. Pri zložitejších problémoch často zjednodušujeme úlohy tak aby boli pre študentov zrozumiteľnejšie avšak pri snahe o čiastkové riešenia oberáme študentov o celkový pohľad na daný problém a možnosť použitia všetkých výhod objektového programovania na riešenie komplexného problému.[1]

Posledným problémom je sústredenie sa na použitý jazyk viac ako na technológiu a celkový prehľad jej možností. Pri snahe čo najviac priblížiť vybraný jazyk študentom školitelia venujú príliš veľa času napríklad základným údajovým typom a riadiacim štruktúram, čo v konečnom dôsledku okresáva čas strávený výučbou princípov objektovo orientovaného programovania.[1]

2.3 Výuka programovania so zameraním na prax

Pri výuke je potrebné aby školiteľ vedel odprezentovať informácie tak, aby si študenti vedeli predstaviť ako by vedeli použiť nadobudnuté znalosti v praxi. Na akademickej pôde je potrebné aby študenti mali teoretické znalosti potrebné na pochopenie technológie, avšak musia vedieť tieto znalosti použiť aj v praxi po ukončení štúdia tak, aby mohli byť okamžite platnými členmi tímu pri vývoji softvéru.[2]

Pri kurzoch, ktoré sú zamerané na začiatočníkov bez predchádzajúcich znalostí je zameranie na čo najväčšiu jednoduchosť a tak sa pokročilejšie témy málokedy spomínajú, čo však vedie k zlým zvykom pri programovaní a začiatočníci sa tak často aj v neskoršej fáze štúdia uchyľujú k zvykom, ktoré nadobudli pri začiatku štúdia. Preto je potrebné študentov už v základoch oboznámiť so všetkými možnosťami paradigmy, ktorá je vyučovaná, nakoľko napríklad študenti často nadobúdajú predstavu, že objektovo orientované programovanie je len o triedach a objektoch a už dedičnosť im spôsobuje bolesť hlavy. O pokročilejších rozhraniach a návrhových vzoroch sa dozvedajú príliš neskoro, kedy už majú vyvinuté nesprávne návyky, ktoré pri návrhu a implementácii už používali dlhší čas a často sa k nim budú uchyľovať ako k jednoduchšej ceste riešenia problémov.[2]

Vyššie spomenuté problémy pri výuke je potrebné eliminovať tak, aby si študent uvedomoval, že v blízkej budúcnosti po nástupe do praxe bude potrebovať nadobudnuté znalosti aby vedel správne komunikovať s kolegami v tíme a taktiež kvôli správnemu použitiu dostupných technológií ktoré mu umožnia postupovať pri vývoji softvéru omnoho rýchlejšie pri implementácii potrieb zákazníka.[2]

3 Výuka programovania na vysokých školách

V tejto časti práce sa pojednáva o vstupných kurzoch programovania na troch zvolených vysokých školách, ktoré používajú princíp postupného vylepšovania programu. Tieto školy boli zvolené kvôli programovaciemu jazyku, v ktorom vyučujú, kvôli typu príkladov, ktoré pri výuke používajú. Jedná sa o dve vysoké školy v Košiciach na Slovensku a o Vysokú školu ekonomickú v Prahe. Následne je uvedené krátke zhrnutie o vstupných kurzoch programovania a analýza námetu postupne vylepšovaných programov.

3.1 TUKE, Technická Univerzita v Košiciach

Pri výuke programovania na Technickej univerzite v Košiciach, ďalej TUKE, sa pri výuke vstupného kurzu v programovaní používa metóda škola hrou. Od roku 2008 prešla výučba na univerzite zmenami v koncepcii vyučovania, kedy sa z cviční vyučujúci rozhodli vypustiť sústredenie sa na technológie a začali sa venovať princípu Objects First.[1]

Počas cvičení sa vyučujúci snažia vyhýbať príkladom, ktoré študentov nikam neposúvajú, namiesto AHA príkladov sa snažia prácou na čiastkových úlohách dopracovať k výslednému programu pomocou postupného osvojovania si znalostí. Cvičenia sú rozdelené do dvoch celkov. V prvom si študenti osvojujú princípy objektovo orientovaného programovania za pomoci prostredia BlueJ. Študenti sa naučia postupne vytvárať prvé triedy a metódy, následne sa naučia pracovať s viditeľnosťou, preťažovaním až sa dopracujú k pochopeniu abstraktných tried a dedičnosti. Študenti si vždy na nasledujúcom cvičení môžu stiahnuť výsledný kód predchádzajúceho cvičenia na ktorom ďalej pracujú a môžu si tak porovnať svoje výsledky s požadovaným správaním. Pre šikovnejších študentov sú taktiež pripravené dodatkové úlohy, ktoré poskytujú ďalší priestor na precvičenie dosiahnutých znalostí.[1]

Námetom tejto časti je obsluha výťahu. Na začiatku študenti vytvoria len jednoduchú triedu predstavujúcu výťah, grafické vyobrazenie je obyčajný obdĺžnik. Na každom cvičení následne pokračujú a doplňujú nové prvky do svojho programu. Je možné pokračovať vo vlastnom programe alebo stiahnuť novú verziu, ktorá odpovedá stavu v akom sa má projekt nachádzať. Študenti v priebehu ďalších cvičení pridávajú pohyb výťahu, jeho ovládanie z kabínky či ovládanie z jednotlivých poschodí.[1]

Druhá ucelená časť sa venuje prípadovej štúdii s použitím prostredia NetBeans. Študenti v tejto časti už nemajú k nahliadnutiu tak podrobné scenáre ako v predchádzajúcej časti, pretože za prípadovú štúdiu už získavajú hodnotenie. Taktiež im už nie je umožnené stiahnuť si výsledok cvičenia, čo študentov núti riešiť úlohy včas. V tejto

časti študenti taktiež pracujú na vylepšovaní svojho projektu od cvičenia k cvičeniu, rozdielom však je, že nie je možné s niečím porovnať priebežný program. Pripravené scenáre predstavujú už len isté postrčenie smerom k príprave celého riešenia podľa vlastných vedomostí. Rozsah celého kurzu je 12 týždňov pričom v každom týždni majú študenti k dispozícii 90 minútové cvičenie. Výučba prebieha hlavne v jazyku Java na ktorom sa demonštrujú úlohy z cvičení.[1]

3.2 UPJŠ, Univerzita Pavla Jozefa Šafárika v Košiciach

Na Univerzite Pavla Jozefa Šafárika v Košiciach, ďalej UPJŠ, prebehla zmena programovacieho jazyka, v ktorom sa programovanie vyučuje v roku 2007 na jazyk Java z pôvodného Pascalu. Pri príprave úvodného kurzu bolo potrebné zohľadniť, že väčšina študentov nebola oboznámená s programovaním v akejkoľvek forme. Z toho vyplýva, že bolo potrebné motivovať študentov tak, aby si objektovo orientované programovanie obľúbili. Úvodný kurz tak možno charakterizovať ako realizáciu prístupu Objektové programovanie najskôr. Z pohľadu práce ide o to naučiť študentov vytvárať a používať objekty. Tak ako na TUKE je kurz rozdelený na dve časti.[14]

V prvej časti sa študenti pomocou korytnačej grafiky naučia základným princípom programovania ako sú premenné, podmienky, cykly a polia. Študenti sa taktiež naučia základy algoritmizácie a neskôr aj vytváranie jednoduchých tried, objektov a metód. V tejto časti sa používa aplikačný rámec JPAZ2. Táto časť kurzu trvá 6 týždňov. Študenti síce nevytvárajú nový projekt, no na každej hodine pridávajú nové znalosti do svojej práce. Na začiatku dokážu vykresliť iba základné útvary ako kruh či obdĺžnik. Študenti postupne vytvárajú nové metódy, ktoré sú vylepšenou formou predošlých metód za použitím nových vedomostí. Úlohy, ktoré na cvičeniach precvičujú sú však na seba nenaviazané.[14]

V druhej časti kurzu, rovnako trvajúca 6-7 týždňov, sa študenti už naučia pracovať so súbormi, naučia sa spracovať výnimky a taktiež sa naučia pokročilé princípy objektovo orientovaného programovania.[14]

Pri príprave kurzu bolo zvažované prostredie BlueJ, avšak po dôkladnom zvážení sa vyučujúci priklonili k prostrediu Eclipse. Taktiež Java bola použitá z dôvodu jednoduchej demonštráciu vybratých príkladov ako aj z dôvodu uplatnenia v praxi.[14]

3.3 VŠE, Vysoká Škola Ekonomická v Prahe

Na Vysokej škole ekonomickej v Prahe, ďalej VŠE, mali študenti na výber medzi dvoma úvodnými kurzmi programovania. Kurz programovania vo Visual Basic sa však už nenachádza v študijnom pláne a tak ostal len kurz programovania v Jave. Tento kurz poskytuje lepší základ pre nadväzný kurz softvérového inžinierstva, keďže

sa v ňom pracuje na vylepšovaní semestrálnej práce z kurzu Javy. V základnom kurze sa využíva prostredie BlueJ a v nadväznom prostredie NetBeans, oba kurzy trvajú 13 týždňov.

Príklady, na ktorých sa študenti učia používať nadobudnuté znalosti sa u vyučujúcich môžu líšiť, cieľom tejto práce je práve navrhnúť možnú variantu týchto príkladov. Ako súčasný príklad cvičení sa môže uviesť práca na projekte autíčok, podľa návrhu pána profesora Ing. Pecinovského v jeho knihe OOP - Learn Object Oriented Thinking & Programming[15]. Okrem práce na týchto príkladoch študenti vytvárajú aj semestrálnu prácu na tému adventúry. Študenti dostanú základnú kostru programu a následne do nej dopĺňajú nové metódy a nové triedy podľa požadovaných prvkov, ktoré má ich program spĺňať.

V kurze softvérového inžinierstva sa následne táto semestrálna práca opäť vylepšuje. Okrem toho však študenti pracujú na tímovom projekte, v ktorom sa využívajú pokročilejšie znalosti ako napríklad práca s databázami.

3.4 Zhrnutie o úvodných kurzoch programovania

Zo zvolených vysokých škôl všetky poskytujú kurz programovania práve v jazyku Java, čo ukazuje o jeho vhodnosti pre výuku pre začiatočníkov. TUKE a VŠE vyučujú programovanie v prostrediach BlueJ a NetBeans zatiaľ čo UPJŠ vyučuje v prostredí Eclipse.

Aj keď kurz na UPJŠ zdokonaľuje program počas výuky, študenti pracujú počas cvičení vždy na úlohách, ktoré sú od seba nezávislé. Toto sa teda líši od kurzu VŠE a TUKE kde študenti zdokonaľujú svoj program počas celého priebehu výuky.

Námet hry však používa zo zvolených škôl len VŠE, TUKE sa tiež stavia k programovania v štýle škola hrou, pre upútanie záujmu študentov, no samotný námet obsluhy výťahu sa ťažko dá pokladať za hru.

3.5 Analýza postupne vylepšovaných programov

Zo vzoru uvedených vysokých škôl ako príklad, je možné predstaviť si rôznorodosť vstupných kurzov. Navyše aj na rôznych kurzoch, v rámci jednej školy, je možné aby vyučujúci používal spôsob výuky, ktorý pokladá za najlepší pre študenta. Nie je teda možné povedať, ktorá metóda výuky je lepšia ako iná.

Výhodou postupne vylepšovaných programov je pre študenta nadhľad, ktorý získa tým, že pracuje dlhšiu dobu len s jedným kódom. Študent na začiatku môže nadobudnúť pocit, že sa v kóde nevyzná a je v ňom stratený, keďže sa pri postupne vylepšovanom programe jedná väčšinou o väčší kus kódu ako pri malých príkladoch môže byť

pocit stratenia väčší. Tento pocit, aj keď možno menší, však nadobudne vždy keď dostane nový kód, v ktorom má pracovať pri samostatných príkladoch. Keďže s postupne vylepšovaným projektom pracuje na každom cvičení, nadobudne tento pocit len raz v priebehu celej výuky.

V rámci prípravy študenta na použitie programovania v praxi je typ postupne vylepšovaných programov tiež miernou výhodou. Študenti pracujú s rozsiahlejším projektom, ktorý by mali udržať v prehľadnom stave aby sa v ňom dokázali orientovať aj po vlastnej úprave. Práca s týmto typom príkladu ich dokáže pripraviť na rozsiahlejšie projekty, s ktorými sa počas praxe môže stretnúť. Tento bod však nepovažujem za zásadný keďže zadania v praxi môžu byť podobnejšie rôznym menším príkladom, ktoré medzi sebou nenesú veľkú podobnosť.

Výhoda jedinej témy príkladu ako motivovanie študenta a upútanie jeho záujmu je otázna. U každého študenta sa zrejme jeho názor na to čo považuje za zaujímavé bude líšiť. Niektorí študenti by považovali prácu na jednom projekte po dlhšiu dobu ako otravnú aj keď by sa jednalo o tému hry, ktorú by samotnú mohli pokladať za zaujímavú. Na opačnej strane zasa existujú študenti, ktorým by neustále preskakovanie z príkladu do príkladu, mohlo pripadať mätúce a teda by sa poznatky, ktoré si z cvičenia odnesú nerovnali poznatkom, ktoré by reálne mali mať.

Ako hlavný problém postupne vylepšovaných problémov by bolo možné uviesť pomernú zlú flexibilitu voči zmenám. Program musí mať pevne stanovený sled krokov, ktoré je nutné postupne prechádzať. Už len samotná stavba príkladu tak aby študenti pridávali do projektu na cvičení to čo sa naučili je zložitá. Navyše je tento typ príkladu pevne zviazaný s obsahom prednášok a teda je viac-menej nepoužiteľný pre iné kurzy. Ako príklad sa dá uviesť vymenenie poradia tém: cykly a podmienky. Ak je program stavaný tak, že študent najskôr tvorí jednoduché cykly a v nasledujúcom cvičení by tieto cykly mal použiť v podmienkach, tak v prípade, že študenti najskôr preberú tému podmienok, nedokázali by pracovať na cvičení, ktoré bolo naplánované a nasledujúce cvičenie by obsahovalo látku, ktorú nepoznajú. Tento problém sa s AHA-príkladmi nevyskytne, keďže príklady sú na sebe nezávislé a teda je možné vybrať príklad podľa preberanej témy.

Z uvedených bodov je možné vidieť, že postupne vylepšované programy sú vhodné jedine pre kurzy s pevne daným rozvrhom tém keďže príklad je postavený na tomto rozvrhu. Keďže sa v tejto práci tvorí príklad pre vstupné kurzy programovania je tento bod splnený. Študenti sa na tomto príklade naučia správnym postupom a návykom. Navyše téma hry by mala byť schopná vytvoriť potrebný záujem u študenta. Vytvorenie postupne vylepšovanej hry pre kurzy programovania je teda dobrý spôsob ako študentom poskytnúť znalosti potrebné pre dokončenie kurzu aj prax spôsobom, ktorý by im pripadal zaujímavý.

4 Programovacie jazyky vhodné pre začiatočníkov

Pri výučbe programovania sa často používajú procedurálne programovacie jazyky ako Pascal alebo pôvodné C. V súčasnosti sa však procedurálna paradigma používa len minimálne a preto je potrebné pri výuke prihliadať aj na súčasnú situáciu pri tvorbe softvéru kde sa najviac používajú objektovo orientované programovacie jazyky. Medzi najpopulárnejšie jazyky v súčasnosti patria JavaScript, Java, Python, C++ a C#. Pre ďalší postup v tejto práci je potrebné definovať aké požiadavky na programovací jazyk máme.

V prípade výučby pre začiatočníkov JavaScript nie je úplne vhodný jazyk, nakoľko je to skriptovací jazyk, ktorý je interpretovaný počas behu programu čo začiatočníkom môže sťažiť odhaľovanie chýb v kóde, pričom použitie objektov v danom jazyku nie je na rovnakej úrovni ako v iných vyšších programovacích jazykoch.[4]

V prípade Pythonu zavažuje proti jeho použitiu vo výučbe to, že je to multi-paradigmový programovací jazyk, čo znamená že umožňuje programátorom používať rôzne prístupy k programovaniu čo v prípade zamerania sa na objektovo orientované programovanie nie je ideálny scenár s prihliadnutím na to, že študenti bez predchádzajúcich skúseností by sa mohli stratiť v možnostiach, ktoré im daný jazyk ponúka. Keďže pri výučbe by sme sa chceli zamerať na prenositeľnosť a multiplatformový prístup jazyk C# nie je vhodný, nakoľko je silne naviazaný na platformu Microsoft Windows. Pri výučbe objektovo orientovaného programovania teda považujeme za najvhodnejšie jazyky Java a C++. Obidva poskytujú objektovo orientovaný prístup, sú multiplatformové a existuje veľké množstvo nástrojov, ktoré poskytujú vyšší používateľský komfort pri písaní zdrojového kódu.[3]

4.1 Programovací jazyk C++

Objektovo orientovaný jazyk C++ vychádza zo staršieho jazyka C čím vývojári reagovali na zastaranosť pôvodného jazyka, ktorý už nespĺňal potreby programátorov a čím viac používaného objektovo orientovaného prístupu. C++ obsahuje všetky potrebné súčasti potrebné pre výučbu objektovo orientovaného programovania, avšak pre úplných začiatočníkov je jeho syntax komplikovanejšia ako v prípade Javy. Taktiež neobsahuje garbage collector, čím je na začínajúcich programátorov vyvíjaný väčší tlak na správu pamäte a životného cyklu objektov čo je na prvý pohľad výhoda, keď programátor má kontrolu nad životnosťou objektov, avšak v prípade začiatočníkov je to zbytočná prekážka, ktorá predstavuje ďalšiu výzvu pri už tak náročnom štúdiu programovania, preto nakoniec pri tvorbe tejto bakalárskej práce padla voľba na použiť jazyk Java.[5]

4.2 Programovací jazyk JAVA

Jazyk Java vznikol v 90-tych rokoch 20. storočia v spoločnosti Sun Microsystems, kde vznikla myšlienka na vytvorenie multiplatformového programovacieho jazyka.[6] Programovací jazyk prešiel búrlivým vývojom a v súčasnosti existujú jeho 3 základné verzie. Java2ME ktorá sa v minulosti používala pre vytváranie aplikácií na mobilné telefóny, Java2SE pre štandardné desktopové aplikácie a Java2EE pre vývoj webových aplikácií. V súčasnosti patrí Java spoločnosti Oracle ktorá ju získala pri akvizícii spoločnosti Sun Microsystems v roku 2009. Posledná verzia 1.9 bola vydaná v roku 2017.[8]

V súčasnosti sa Java používa hlavne vo webových aplikáciách a na programovanie aplikácií na platformu Android. Pri výučbe sa používa hlavne pre svoju multiplatformovosť a taktiež, z dôvodu jej voľnej dostupnosti. Taktiež existuje aj open source variant OpenJDK, ktorý pokračuje v snahe spoločnosti Sun v roku 2006 o open source implementáciu tejto technológie pod GNU licenciou.[7]

5 Integrované vývojové prostredia

Pri tvorbe programu programátor nepotrebuje vývojové prostredie. Pre programovanie niektorým programátorom postačuje textový editor a kompilátor, avšak pre efektívnejšiu a precíznejšiu prácu s kódom v priebehu rokov vzniklo množstvo vývojových prostredí. V prípade Javy potrebujeme na správne programovanie dve súčasti. Jednou je Java SDK čo je súbor nástrojov, ktoré umožňujú vytváranie softvéru pre platformu Java. Ďalšou súčasťou je Java RE čo je prostredie na beh aplikácií, v ktorom ,ako názov napovedá, sa naše aplikácie spúšťajú, teda umožňuje spúšťanie inštrukcií v poradí v akom to určil programátor pri vývoji softvéru.

Ak má programátor všetky tieto súčasti v svojom zariadení môže vytvárať aplikácie pre platformu Java. Avšak ako bolo spomenuté pre efektívnejší vývoj je vhodné použiť niektoré vývojové prostredie. Pre platformu Java existuje veľké množstvo takýchto prostredí. Medzi najpopulárnejšie patria Eclipse, IntelliJ a NetBeans.

5.1 Vývojové prostredie Eclipse

Prostredie Eclipse pôvodne vzniklo ako proprietárna technológia v dcérskej spoločnosti IBM. Cieľom tohto prostredia bolo pôvodne zjednotiť veľké množstvo nekompatibilných prostredí, ktoré používali zákazníci a tým zvýšiť možnosti opätovného použitia spoločných prvkov týchto prostredí. Toto prostredie nevzniklo úplne od nuly ale z už existujúcich prostredí napísaných v SmallTallku. V roku 2001 bol Eclipse oznámený ako open source projekt skupinou spoločností, ktoré stvorili počiatočné Eclipse Consortium.[9]

Platforma Eclipse je univerzálnym nástrojom, v ktorom je vďaka pluginom možné programovať v akomkoľvek jazyku. Prostredie samo o sebe neposkytuje veľa možností, ale poskytuje možnosti rozšírenia, ktoré z neho činia nástroj v ktorom môže používateľ pracovať na rôznych projektoch.[9]

V prostredí Eclipse je základným prvkom takzvané pracovné prostredie(workbench) v ktorom sa nachádza adresárová hierarchia, ktorá obsahuje napríklad zdrojové kódy, projekty a stavové informácie o spustenom plugine. Každé workbench okno poskytuje perspektívy, ktoré môžu obsahovať jeden alebo viac pohľadov a editory na správu zdrojového kódu.[9]

5.2 Vývojové prostredie IntelliJ

IntelliJ je v súčasnosti najpopulárnejšie vývojové prostredie pre programovanie v jazyku Java. Prostredie je vyvíjané spoločnosťou JetBrains a je distribuované pod Apache2 komunitnou licenciou a taktiež pod proprietárnou licenciou, pričom v obidvoch prípadoch môže byť použité pre komerčné účely.[10]

Prostredie má hlboký pohľad na kód a taktiež aj kontext napísaného kódu čo z neho činí unikátne prostredie medzi ostatnými integrovanými prostrediami pre Javu. Medzi jeho hlavné prednosti patrí inteligentné doplňovanie kódu, vyhľadávanie duplikátov, inšpekcia a rýchle fixovanie pričom všetky tieto možnosti sú zaznamenávané prostredím samotným a tým pádom umožňujú efektívnejšie editovanie kódu.[10]

Prostredie je vybudované tak, aby umožňovalo písanie kódu s minimom rozptyľovania, tým pádom je vždy viditeľný na obrazovke len editor kódu pričom má pridelené skratky pre ostatné funkcie, ktoré nesúvisia so samotným písaním kódu.[10]

Pre pomoc programátor so zjednodušením procesu tvorby kvalitného kódu prostredie má vstavané programátorské nástroje ako napríklad prepojenie s verzovacími systémami ako GitHub, CVS, Perforce a TFS. Taktiež poskytuje nástroje na zostavenie kódu, testovacie prostredie, terminál, databázové nástroje, aplikačný server a podporu pre Docker .[10]

IntelliJ sa distribuuje v komunitnej a ultimátnej verzii. Pričom komunitná verzia sa používa najmä pri vývoji pre JVM a Android a druhá verzia je zameraná na vývoj webových a korporátnych aplikácií. Ultimátna edícia je distribuovaná pod Apache 2 licenciou a nepodporuje prepojenie s Perforce, TFS nepodporuje taktiež JavaScript, TypeScript, prepojenie na databázu a databázové nástroje a taktiež rôzne Java frameworky.[10]

6 Úvod do hier

Praktická časť tejto práce sa venuje návrhu počítačovej hry v jazyku Java ako pomôcku pre vstupný kurz programovania. Keďže tento návrh nie je zvolený, len pre zaujatie študentov, ale aj pre jeho možnosti využitia v praxi, je vhodné uviesť príklady tohto využitia a taktiež sa pozrieť na editory, ktoré umožňujú programátorom rýchlejší a intuitívnejší vývoj hier.

6.1 Java hry

V minulosti boli Java hry dostupné najmä na mobilných telefónoch. Prvé hry sa distribuovali ešte na mobilné telefóny bez farebnej obrazovky v roku 2002. V tej dobe to bola veľká novinka, ktorá definovala hranie hier na mobilných zariadeniach až do príchodu inteligentných telefónov. Pre vývoj hier pre mobilné zariadenia bola používaná Jave2ME špeciálne navrhnutá práve pre mobilné zariadenia a ich obmedzené hardvérové zdroje.

Najznámejšie tituly , ktoré boli vytvorené prostredníctvom programovacieho jazyka Java sú pravdepodobne hry zamerané na mobilné platformy ako napríklad Angry Birds, Fruit Ninja alebo God of War Betrayal.

V prípade počítačových hier Java nie je veľmi používaný programovací jazyk. Avšak jedna z najznámejších hier posledných rokov Minecraft je naprogramovaná práve v jazyku Java. Táto hra podporuje predstavivosť hráča a umožňuje v kreatívnom móde budovanie čohokoľvek čo si hráč predstavuje, pričom v tomto móde hráč nie je limitovaný fyzikálnymi zákonmi a má neobmedzené množstvo zdrojov. V prípade módu o prežitie má taktiež neobmedzené možnosti avšak hráč musí získavať bloky potrebné na stavbu vlastným úsilím a podľa zvolenej náročnosti musí bojovať aj proti príšerám alebo s inými hráčmi. Hra sa nesnaží zaujať grafickým spracovaním ale svojimi možnosťami a prekážkami čím podporuje hráčovu predstavivosť a myslenie.[11]

6.2 Editory pre vývoj počítačových hier

Pri vývoji hier môžu programátori používať pokročilejšie editory, ktoré sú dostupné zdarma pre nekomerčné použitie a na ktorých vznikajú známe aj menej známe hry a ponúkajú obrovské možnosti pre tvorbu pokročilejších hier. Najznámejšie sú Unreal Engine a Unity.

6.2.1 Unreal Engine

Unreal Engine je kompletný set, vývojárskych nástrojov, navrhnutých tak aby stretla očakávania ambicióznych výtvarných vízií, zatiaľ čo zostáva dostatočne flexibilná pre dosiahnutie úspechu pre tímy všetkých veľkostí. Unreal Engine je produkt, ktorý je pripravený na produkciu hneď po doručení bez nutnosti vkladania ďalších pluginov alebo ďalšieho dokupovania. Umožňuje vytvoriť hrateľný obsah bez nutnosti napísania jedinej čiarky kódu, pomocou metódy Blueprints. Má dostupný celý zdrojový kód, ktorý môže užívateľ voľne študovať a prispôsobovať si ho úplne bez obmedzení. Poskytuje nástroj na úpravu animácií a kinematiky na profesionálnej úrovni. Animácie vytvorené pomocou tohto enginu sú ako zo skutočného sveta. Unreal Engine navyše podporuje tvorbu na rôzne platformy najnovšie konzoly od Xboxu, Playstation či Nintenda, ale aj na PC a mobily.[12]

6.2.2 Unity

Všetko v jednom editor, ktorý sa rozširuje aby spĺňal kritéria zákazníka. Je dostupný ako na Windows tak aj na Mac, ponúka široké spektrum nástrojov pre výtvarné potreby na navrhovanie herných svetov a pre zdokonaľovanie schopností, ale aj silnú sadu vývojárskych nástrojov na implementáciu hernej logiky a hrateľnosti. Podporuje tvorbu 2D aj 3D, s funkcionalitou a doplnkami pre špecifické potreby užívateľa naprieč rôznymi žánrami. Vysoko realistická a vysoko výkonná hrateľnosť vďaka fyzikálnym prvkom od Box2D a NVIDIA PhysX. Editor sa dá rozširovať, ktorýmkoľvek nástrojom potrebným pre prácu tímu. Možnosť vytvorenia si vlastných doplnkov alebo nájdenia si doplnkov na stiahnutie z tisícov zdrojov pre urýchlenie práce na projekte.[13]

7 Návrhy hier pre praktickú časť

Okrem návrhu, ktorý sa bude spracúvať v praktickej časti práce, sú v tejto časti uvedené ďalšie návrhy, ktoré by bolo možné použiť vo výuke programovania. Návrhy majú opísaný cieľ hry jej priebeh, možné vylepšenia a tiež je uvedené prečo bol respektíve nebol použitý v praktickej časti. Taktiež je uvedené akou hrou alebo hrami bol návrh inšpirovaný.

7.1 Trader

Návrh spočíva vo vytvorení simulácie obchodníka, ktorého cieľom je nadobudnúť čo najväčšie množstvo zlata. Hráč začína svoju púť v náhodnom meste sveta, so základným množstvom zlata a bez žiadneho ďalšieho majetku. Na obrazovke sa zobrazujú akcie, ktoré môže vykonať, ako napríklad nákup či predaj tovaru, získavanie informácií o cenách tovaru v iných mestách alebo opustenie mesta. Hráč následne ovláda svoju postavu klávesnicou aby sa dostal do niektorého z ďalších miest kde sa snaží svoj majetok navyšovať. Možné vylepšenia by mohli byť napríklad nájomní žoldnieri, ktorí by hráča ochraňovali na cestách pred útokmi banditov. Nové spôsoby dopravy a s tým spojené množstvo tovaru, ktoré hráč dokáže prepravovať, poprípade časti mapy, ktoré sú sprístupnené len s nejakou podmienkou ako napríklad potreba lode na dosiahnutie mesta na ostrove. Poprípade by mohli študenti využiť svoje znalosti z ekonomickej sféry ako dane, clo či menový kurz medzi rôznymi druhmi platidiel.

Návrh bol inšpirovaný hlavne flashovou hrou Master of the Secret Sea(Námorne dobrodružstvo) a čiastočne hrami zo série Sid Meier's Civilisation(Civilizácia). Medzi kladné prvky návrhu by sa dali zaradiť, možnosť použitia ekonomických znalostí, keďže sa jedná o študentov VŠE a veľká sloboda vlastného obsahu. Návrh však nebol použitý pre jeho podobnosť s témou semestrálnej práce adventúry, keďže obsahujú podobné prvky ako pohyb medzi rôznymi priestormi hry či manipulovanie s inventárom.

7.2 Dragon Wars

Jedná sa o návrh, ktorý je inšpirovaný klasickými simulátormi boja hlavne mobilných hier ako Monster Legends, Dragon Mania alebo svetoznáma legenda od Nintenda Pokémon. Hráč si na začiatku zvolí jedného draka, ktorý patrí do nejakej kategórie, môže sa jednať o klasický systém oheň, voda, tráva kde oheň poráža trávu, ale prehráva s vodou a voda prehráva s trávou, ale nie je to podmienkou. Hráč následne môže získavať nových drakov, vylepšovať ich a používať ich v súbojoch. Súboje môžu

byť generované náhodne alebo si študenti môžu pripraviť scenár podľa, ktorého budú nepriatelia vyberaní.

Návrh nebol zvolený hlavne z dôvodu dĺžky priebehu samotnej hry. Štýl postupného vylepšovania svojej postavy alebo tímu, nepokladám za vhodný ako návrh pre kurz programovania. Na príklade hry pokémon môžeme vidieť, že príbeh trvá rámcovo niekoľko hodín. Samozrejme záleží od samotného programu, ale nepredpokladám, že študenti by považovali za zaujímavé vytvárať hru, v ktorej by bolo nutné získavať levely aby sa posunuli ďalej v príbehu. Pokiaľ by sa naopak jednalo o skrátenú verziu kde by existovalo len pár súbojov, tak by hra nepredstavovala žiadnu zábavu. Prečo bol návrh zvažovaný bol hlavne dôvod, úplne voľnej ruky v návrhu. Postavičky nemusia byť draky ale vojaci, patriaci do rôznej kategórie. Počet kategórií a typy kategórií sú tiež voľne voliteľné, nemusí ísť o vodu, oheň, trávu ale napríklad o klasickú RPG zostavu bojovník, mág, zlodej a podobne.

7.3 Slalom

Ako názov napovedá jedná sa o jednoduchú hru predstavujúcu lyžiarsky slalom. Úlohou hráča je rovnako ako v skutočnom slalome prechádzať pomedzi bránky, až nakoniec dôjsť do cieľa. Hra by bola ovládaná klávesnicou kde by študent ovládal smer, v ktorom sa má hráč pohybovať, vpravo alebo vľavo, aby sa trafil do bránky. Možným vylepšením by bola určitá kontrola rýchlosti, ktorá by ovplyvňovala schopnosť zatáčať. Vo vysokej rýchlosti by samozrejme hráč nedokázal vytáčať ostré zákruty, alebo by takáto snaha vyvrcholila pádom. Ďalšou možnosťou by mohli byť určité prekážky na trati. Ľadová vrstva, na ktorej sa nedá zatáčať alebo kopčeky, ktoré by vystrelili hráča dopredu podľa rýchlosti a podobne. Opäť by mohlo ísť o vopred pripravenú trasu alebo náhodne generovanú trasu, ktorá by len spĺňala určité podmienky aby nenastala situácia kde by bolo dokončenie hry úplne nemožné.

Podobný návrh predstavuje napríklad hra Rolling Sky, v ktorej je úlohou hráča navigovať loptičku po platniach bez toho aby spadla do priepasti alebo narazila do neprechodnej prekážky. Platne sa môžu prepadávať poprípade vystreľovať loptičku naprieč levelom, prekážky môžu byť pohyblivé alebo reagovať na rôzne spínače. Návrh považujem za zaujímavý a zábavný pre študentov, hra nepotrebuje žiadne taktické a zdĺhavé prípravy. Poprípade je možné vytvoriť paralelný slalom kde by zároveň proti sebe pretekali dvaja študenti na rovnako postavenej trati, alebo dokonca na tej istej trati. Návrh nebol zvolený kvôli podobnosti z návrhom autíčok, ktorý sa v úvodnom kurze už používa.

7.4 Elemental

Vzorom hry by sa dala označiť kombinácia hier Mario a Pokémon od Nintenda a Dino Runner, hra ktorú je možné hrať na prehliadači Google Chrome v off-line režime. Cieľom hry je dostať sa čo najďalej bez prehry. Hráč ovláda svoju postavičku a stretáva náhodne generované monštrá. Monštrá patria do jednej z troch kategórií oheň, voda, tráva. Hráč dokáže svoju postavičku premeniť podľa potreby na príslušníka jednej z týchto kategórií alebo na obyčajnú kategóriu. Ako v hre Pokémon je zachovaná prevaha jednotlivých elementov. Obyčajná kategória spôsobuje všetkým monštrám rovnakú stratu zdravia, ale táto hodnota je prirodzene nižšia ako použitie dominantného elementu. Hráč nemusí nepriateľov zničiť ale len preskočiť to sa však odrazí na množstve bodov, ktoré získa.

8 Praktická časť

8.1 Zámer

Študenti na začiatku obdržia jednoduchý základ programu, na ktorom následne budú pracovať aby vytvorili konečnú verziu hry. V ďalšej časti práce je popísané čo majú študenti naprogramovať počas jednotlivých častí práce. Jedná sa teda o návod vytvorenia jednoduchej hry. Od študentov sa pred každou úlohou očakáva istá znalosť programovacieho jazyka Java. Tieto predpoklady sú vždy uvedené na začiatku jednotlivých cvičení. Vždy sa očakáva iba čiastočný pokrok vo vedomosti a cvičenia sú rozdelené na zvládnuteľné časti. Jednotlivé cvičenia sú pripravené tak aby bolo možné dokončiť cvičenie v čase jednej hodiny. Študenti ktorí majú už nejakú skúsenosť s programovaním to samozrejme zvládnu rýchlejšie. No nemala by nastať situácia kedy by niekto nebol schopný dokončiť prácu počas výuky. Môj projekt bude tvorený v prostredí Eclipse, nie je to však podmienkou.

8.2 Cvičenie 1:

8.2.1 Predpoklady k cvičeniu:

8.2.2 Úloha:

V priebehu prvého cvičenia študenti dostanú základný balíček. V tejto časti sa neočakáva žiadna znalosť programovania. Toto cvičenie je len na úvod do projektu a v podstate sa projekt nijak neposunie vpred. Účelom je zoznámenie sa s prostredím a so základným princípom projektu.

Je možné prezrieť si štruktúru svojho projektu a je možné spustiť Launcher. V novom okne sa následne spustí doterajší kód. Keďže doňho zatiaľ nič nebolo pridané uvidia všetci tú istú scénu. Prázdny obraz otvorený v novom okne.

Práve do tohto plátna budeme počas projektu zakresľovať obraz a udalosti našej hry. Textúry, ktoré použijeme sa nachádzajú v základnom balíčku v priečinku resources, textures.

Všetky textúry použité v projekte sú súčasťou základného balíčka, a je samozrejme možné upraviť ich podľa vlastných preferencií, hra tak získa jedinečnosť a výsledný program bude aspoň graficky unikátny. Táto úprava je vhodná na prvé cvičenie pri dostatku času a bez samotného programovania. Predpripravené textúry sú z väčšej časti vlastná tvorba. Na textúru podlahy budú použité textúry z hry Minecraft. Ako textúra elementov je použitý vzor z hracích kariet Pokémon.

Súčasťou základného balíčku sú triedy zaradené do štyroch balíkov.

Balík e_display obsahujúci triedu GameScene. S touto triedou sa počas celého projektu nebude pracovať. Jedná sa o konštruktor ktorý vytvorí JFrame a Canvas, teda plátno, na ktoré sa počas celého projektu budeme „kresliť“. Obsahuje metódy getFrame, getCanvas a createScene.

Ďalej balík e_game, ktorý sa stará o priebeh celej hry. Triedy v balíku sú Launcher a Game. Trieda Launcher, ktorá sa spúšťa obsahuje len metódu Main, ktorá inicializuje samotnú hru, špecifikuje sa šírka, výška a titulok použité pri vytvorení JFramu. Trieda Game zakladá scénu pomocou parametrov, ktoré dostala pri svojej inicializácií triedou Launcher. Metóda game implementuje triedu Runnable, čím dostáva povinnú metódu run. Neustály chod programu je zabezpečený práve v metóde run, v ktorej sa neustále opakuje volanie metód update a render. Tieto dve metódy sú dôležité lebo informujú program o tom čo sa má udiať a vykresliť na plátno, tieto metódy sa vyskytnú naprieč celým projektom.

Balík e_graphics. Triedy ImageLoader a Assets, trieda ImageLoader obsahuje len konštruktor, ktorý vracia obrázok typu BufferedImage na základe parametru, ktorým je cesta k súboru. Assets je trieda v ktorej sa všetky textúry nahrajú do premenných, tieto premenné sú prístupné naprieč celým projektom.

Balík e_states, obsahujúci v základnej verzii len triedy State a GameState. Trieda State je abstraktnou triedou, určuje povinné metódy pre triedy, ktoré ju ďalej rozširujú ako je aj trieda GameState. Táto trieda určuje stav aplikácie, v ktorom beží hra.

8.2.3 Výsledok cvičenia:

Študenti si založili svoj projekt do prostredia Eclipse, zoznámili sa s prostredím a mierne sa zoznámili aj so samotným projektom. Vyskúšali si prvé spustenie, a mali príležitosť pripraviť si vlastné textúry, ktoré by používali ďalej v projekte.

8.3 Cvičenie 2:

8.3.1 Predpoklady k cvičeniu:

balík, trieda, premenná, typy premenných, parameter, this, metódy

8.3.2 Úloha:

Vytvorenie novej triedy Level a novej triedy Manager. V triede Level bude prebiehať hlavné renderovanie celého projektu. Budú sa tu vykresľovať objekty v hre ako hráč či nepriatelia, pozadie celej hry, dlaždice tvoriace podlahu a ďalšie. Trieda Manager bude mať jedinú úlohu a to distribuovať inštancie tried Game a tried Level. Keď budeme niekde v projekte potrebovať pracovať s triedami Level alebo Game budeme používať práve triedu Manager.

Prvým krokom bude vytvorenie balíku e_level. V tomto balíku následne vytvoríme samotnú triedu Level.

V našej novo vytvorenej triede ďalej vytvoríme dve metódy, prvou bude metóda update bez parametru a druhou metóda render, ktorá bude ako parameter potrebovať inštanciu Graphics, pre inšpiráciu sa môžeme pozrieť do triedy Game kde metódy update a render máme taktiež. Na to aby sme ale triedu Graphics mohli použiť musíme importovať balíček, ktorý túto triedu obsahuje. Použijeme nasledovný import.

```
import java.awt.Graphics;
```

Aby naša trieda nebola úplne prázdna a aby sme upravili prázdne okno, ktoré momentálny program spúšťa, vykreslíme na naše plátno pozadie našej hry. Keďže kreslíme použijeme metódu render. V nej použijeme nasledovný príkaz.

```
graphics.drawImage(Assets.background, 0, 0, null);
```

V tomto príkaze graphics označuje inštanciu Triedy Graphics, ktorú sme zadali ako parameter našej metódy render, drawImage je metóda pochádzajúca z tejto triedy. Parametre tejto metódy sú následne obrázok, ktorý sa má vykresliť, ten dostaneme z predpripravenej triedy Assets, ďalej dva celočíselné parametre označujúce súradnice odkiaľ sa má obrázok vykresliť. A posledným parametrom je observer, s tým však v našom projekte nebudeme pracovať a teda vždy bude hodnotou null.

Toto zatiaľ stačí v triede Level, samozrejme je možné vykresliť si aj iné obrázky nie len pozadie, no to je už na študentoch.

Aby však naše pozadie bolo vykreslené potrebujeme metódu render z triedy Level niekde zavolať. Toto zavolanie prebehne v triede GameState. Táto trieda predstavuje stav projektu keď beží naša hra.

Najskôr si vytvoríme súkromnú premennú inštancie triedy Level.

```
private Level level;
```

Túto premennú nainicializujeme v konštruktore triedy GameState.

```
level = new Level();
```

Nakoniec zavoláme metódu render z triedy Level. Toto zavolanie prebehne vo vnútri metódy render triedy GameState.

```
public void render(Graphics graphics) {
level.render(graphics);
}
```

V metóde render triedy GameState s parametrom Graphics graphics, čiže inštancia triedy Graphics, zavoláme metódu render z triedy Level. Metódu render voláme pomocou vytvorenej súkromnej premennej, ktorá predstavuje inštanciu triedy Level. Táto metóda potrebuje ako parameter inštanciu triedy Graphics a tak použijeme parameter z metódy render v triede GameState.

Keď následne spustíme program, na naše plátno sa vykreslí pozadie.

Druhou časťou cvičenia bude vytvorenie triedy Manager.

Túto triedu vytvoríme v balíku e_game.

Keďže bude pracovať s triedami Level a Game, vytvoríme si inštančné premenné týchto dvoch tried.

```
private Game game;
private Level level;
```

Opäť je nutné importovať balíček tentokrát však importujeme náš vlastný balíček.

```
import e_level.Level;
```

Následne vytvoríme metódy getWidth, getHeight, getGame, setGame, getLevel a setLevel. Tieto metódy sa nazývajú gettre a settre.

```
public int getWidth() {
return game.getWidth();
}

public int getHeight() {
return game.getHeight();
}

public Game getGame() {
return game;
}

public void setGame(Game game) {
this.game = game;
}

public Level getLevel() {
return level;
}

public void setLevel(Level level) {
this.level = level;
}
}
```

Aby sme túto triedu mohli používať musíme ju niekde nainicializovať. To urobíme v triede game.

Opäť najskôr súkromná premenná.

```
private Manager manager;
```

A následne v metóde init(), inicializácia.

```
manager = new Manager();
```

Aby sme však mohli distribuovať inštanciu triedy Game musíme ju priradiť premennej v triede Manager. Na to použijeme setter, ktorý sme si vytvorili setGame.

```
manager.setGame(this);
```

Môžeme si všimnúť, že parameter metódy je this. This označuje triedu, v ktorej sa momentálne nachádzame respektíve triedu, v ktorej sa this použije.

Aby sme si ukázali ako to vlastne bude fungovať, presunieme sa do triedy State. V nej si môžeme všimnúť premennú.

```
protected Game game;
```

Protected znamená, že túto premennú zdedí každá trieda, ktorá bude triedu State rozširovať, o tom však až v ďalšom cvičení.

Miesto toho aby sme vytvorili inštanciu Game vytvoríme inštanciu Manager.

```
protected Manager manager;
```

A vo všetkých prípadoch keď trieda State použije premennú game použijeme miesto toho premennú manager. Rovnakú úpravu vykonáme aj v triede GameState. V triede Game ešte zmeníme inicializáciu triedy GameState a miesto parametra this jej dáme parameter manager. Na ukážku čo sa stalo si spustíme príkaz.

```
System.out.println(manager.getWidth());
```

Tento príkaz sa pridá do metódy update v triede GameState. Teraz nám pri každom update po spustení programu, na konzole vyskočí údaj o šírke našej hry, ktorú sme dostali cez triedu Manager. Túto časť kódu môžeme následne zmazať.

Posledným krokom je, že inštanciu triedy Level priradíme do premennej v triede manager. To spravíme v triede GameState kde pridáme do konštruktora metódu setLevel.

```
manager.setLevel(level);
```

Teraz dokážeme distribuovať inštancie tried Level a Game pomocou novej triedy Manager.

8.3.3 Výsledok cvičenia:

Študenti si vytvorili svoje prvé triedy, Level a Manager, svoje prvé metódy, ktoré následne aj zavolali. Trieda Level má za úlohu vykresľovať jednotlivé objekty našej hry. Študenti prvý krát zobrazili pozadie na svojom plátne. Trieda Manager sa stará o poskytovanie inštancií tried Level a Game.

8.4 Cvičenie 3:

8.4.1 Predpoklady k cvičeniu:

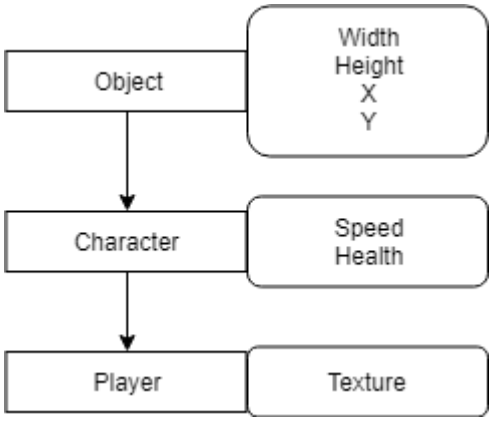
rozširovanie triedy, abstraktné triedy, abstraktné metódy

8.4.2 Úloha:

Toto cvičenie sa opäť bude venovať vytváraniu nových tried. Tento krát sa vyrobí hneď niekoľko tried, ktoré však budú na seba nadväzovať. Študenti si vyobrazia svojho hráča.

Triedy, ktoré sa budú vytvárať, sú Object, Character a Player. Tieto triedy sa budú medzi sebou rozširovať.

Ako je možné vidieť na *obrázku 2* každá z tried pridá niečo nové. Object bude charakterizovaný jeho šírkou, výškou a pozíciou v priestore. Character bude mať taktiež šírku, výšku a pozíciu v priestore, ale k tomu ho ešte charakterizuje rýchlosť pohybu a zdravie. Trieda Player ako by sa dalo očakávať zdedí všetky z charakteristických vlastností triedy Character, ale pridá k tomu samotnú textúru, ktorú vykreslí.



Obrázok 2: Jednoduchý diagram rozširovania tried v projekte

Vytvorenie balíku e_objects.

Vytvorenie triedy Object. Triedu študenti vytvoria do balíku e_objects. Trieda však bude abstraktná. Ako dedené premenné sa vytvoria inštancia triedy Manager, pozície na ploche x, y a šírka s výškou. Premenné x, y môžu byť použité pri výpočtoch, ktoré nevrátia celé číslo a tak by mali byť typu double alebo float, pre tento projekt float postačí.

```
protected Manager manager;
protected float x, y;
protected int width, height;
```

Vytvorenie konštruktora Object. Konštruktor bude mať parametre typu Manager, float, float, int, int. Premenné typu int budu reprezentovať šírku a výšku objektu, zatiaľ čo float hodnoty budú súradnice ako už bolo spomenuté. V konštruktory sa nainicializujú premenné zo zadaných parametrov.

```
public Object(Manager manager, float x, float y, int width, int height) {
this.manager = manager;
this.x = x;
this.y = y;
this.width = width;
```

```
this.height = height;
}
```

Vytvorenie metód triedy Object. Vytvoria sa gettre a settre pre šírku, výšku, a obe súradnice. Abstraktné metódy, ktoré sa použijú budú render a update, metóda render berie vždy ako parameter inštanciu triedy Graphics, študenti sa môžu inšpirovať deklaráciou tejto metódy v triedach Level či Game.

```
public float getX() {
return x;
}

public void setX(float x) {
this.x = x;
}

public float getY() {
return y;
}

public void setY(float y) {
this.y = y;
}

public int getWidth() {
return width;
}

public void setWidth(int width) {
this.width = width;
}

public int getHeight() {
return height;
}

public void setHeight(int height) {
this.height = height;
}

public abstract void update();
public abstract void render(Graphics graphics);
```

Vytvorenie triedy Character. Trieda bude rozširovať triedu Object, ktorá bola vytvorená ako prvá. Opäť sa však bude jednať o abstraktnú triedu. Dedené premenné budú zdravie a rýchlosť. Rýchlosť bude opäť typu float. Okrem nich sa vytvoria ešte verejné stále premenné na základnú rýchlosť a základnú veľkosť zdravia, pre tento projekt bude použitá rýchlosť 6.0f a zdravie 9. Ďalšie verejné stále premenné, ktoré sa vytvoria budú na základnú šírku a základnú výšku postavy, v tomto projekte to budú hodnoty 75 na šírku a 100 na výšku.

```
public static final int defHealth = 9;
public static final float defSpeed = 6.0f;
public static final int defWidth = 75, defHeight = 100;
```

```
protected int healthBar;
protected float speed;
```

Vytvorenie konštruktora triedy Character. Konštruktor triedy, ktorá rozširuje inú triedu je špeciálny, kde sa volá príkaz super, ktorý odkazuje na supertriedu, teda na triedu, ktorú rozširujeme. V konštruktore ešte priradíme premenným health a speed ich základné hodnoty.

```
public Character(Manager manager, float x, float y, int width, int height) {
super(manager, x, y, width, height);
healthBar = defHealth;
speed = defSpeed;
}
```

Vytvorenie metód pre triedu Character. Metódy budú zatiaľ len gettre a settre pre rýchlosť a zdravie.

```
public int getHealthBar() {
return healthBar;
}

public void setHealthBar(int healthBar) {
this.healthBar = healthBar;
}

public float getSpeed() {
return speed;
}

public void setSpeed(float speed) {
this.speed = speed;
}
```

Vytvorenie triedy Player. Trieda bude rozširovať triedu Character. Vytvorí sa jediná súkromná premenná typu BufferedImage.

```
private BufferedImage playerImage;
```

Vytvorenie konštruktora triedy Player. V konštruktory tentokrát vynecháme parametre pre šírku a výšku. To spôsobí chybu, ktorú napravíme tým že v odkaze na supertriedu namiesto pôvodných parametrov šírky a výšky vložíme základnú výšku a šírku, ktorú sme vytvorili v triede Character. Nainicializujeme ešte súkromnú premennú typu BufferedImage, ktorej priradíme hodnotu, niektorej textúry z triedy Assets.

```
public Player(Manager manager, float x, float y) {
super(manager, x, y, defWidth, defHeight);
playerImage = Assets.playerBasic[0];
}
```

Vytvorenie metód triedy Player. Metódy, ktoré sa použijú budú render a update. Keďže sa jedná o dedené metódy z triedy Object, mali by sme pred tieto metódy uviesť @Override, čo naznačuje že sa tieto metódy prepisujú. Do metódy render, vložíme príkaz na vykreslenie obrázku, opäť sa môžeme inšpirovať v triede Level. Ako parametre príkazu uvedieme obrázok, ktorý sme uložili do premennej a súradnice, v tomto prípade použijeme x a y, ktoré budú deklarované pri vytvorení inštancie triedy Player. Keďže sa očakávajú celočíselné parametre, tak pred x a y doplníme (int) čo tieto parametre konvertuje na celé čísla. Posledným parametrom je observer, ale s ním v tomto projekte nebudeme pracovať a tak tam zadáme null.

```
@Override
public void update() {

}

@Override
public void render(Graphics graphics) {
    graphics.drawImage(playerImage, (int) x, (int) y, null);
}
```

Vykreslenie hráča na obrazovku. V triede Level vytvoríme súkromnú premennú typu Player.

```
private Player player;
```

Vytvoríme konštruktor tejto triedy. Ako parameter bude mať inštanciu triedy Manager.

```
public Level(Manager manager) {
```

Tento krok spôsobí chybu pretože v triede GameState kde inicializujeme premennú level, používame konštruktor bez parametra a tak túto inicializáciu upravíme.

```
level = new Level(manager);
```

Našu premennú player následne nadeklarujeme v konštruktore triedy Level. Ako prvý parameter sa použije inštančná premenná typu Manager, ďalšie dva parametre symbolizujú polohu, nám zatiaľ postačí vyplniť obe hodnoty 100.

```
player = new Player(manager, 100, 100);
```

Aby však hráča vykreslilo musí prebehnúť metóda render triedy Player, a tak túto metódu zavoláme v metóde render triedy Level. Pokiaľ túto metódu zavoláme na začiatku, tak to vyzerá ako by nič nevykreslilo, no v skutočnosti nám nášho hráča vykreslilo len ho prekrylo pozadie, ktoré sa vykresľuje následne, preto túto metódu zavoláme až na konci metódy render.

```
player.render(graphics);
```

8.4.3 Výsledok cvičenia:

Študenti vytvorili abstraktné triedy Object a Character a triedu Player, ktorá bude predstavovať nášho hráča. Zasiahli tiež do toho čo sa vyobrazuje na našom plátne pomocou metód render, a mohli si vyskúšať že záleží na poradí, v ktorom jednotlivé prvky vykresľujeme. Okrem toho sa naučili konvertovať float hodnotu na int hodnotu.

8.5 Cvičenie 4:

8.5.1 Predpoklady k cvičeniu:

interface, implementácia interfacu, podmienky, pole

8.5.2 Úloha:

V tomto cvičení sa budeme zaoberať pohybom hráča na základe stláčania klávesov na klávesnici, s tým budú súvisieť podmienky if a switch. Budeme vytvárať triedu implementujúcu interface KeyListener, na základe ktorej budeme reagovať na stlačenie konkrétnej klávesy.

Vytvorenie nového balíku e_input.

Vytvorenie novej triedy KeyInput v balíku e_input. Trieda bude implementovať interface KeyListener. Je nutné implementovať povinné metódy vyplývajúce z interfacu KeyListener. Tieto metódy sú keyPressed, keyReleased a keyTyped. Metóda keyPressed reaguje na stlačenie klávesy, keyReleased na pustenie stlačenej klávesy a keyTyped reaguje len v prípade, že bol niekde zapísaný kód predstavujúci stlačenú klávesu. Ešte pred konštruktorom si vytvoríme pár premenných. Ako prvé si vytvoríme súkromné pole hodnôt kľúčov typu boolean teda pravda alebo nepravda. Následne dve verejné premenné typu boolean pre zaznamenanie smeru vpravo alebo hore, keďže v našej hre budeme potrebovať len pohyb vpravo a skok. V implementovaných metódach si môžeme zmeniť názov parametru pre lepšiu orientáciu a lepší zápis.

```
public class KeyInput implements KeyListener{
private boolean[] keys;
public boolean right, up;

@Override
public void keyPressed(KeyEvent key) {

}

@Override
public void keyReleased(KeyEvent key) {

}

@Override
```

```
public void keyTyped(KeyEvent key) {

}
}
```

Vytvorenie konštruktora triedy KeyInput. Konštruktor nebude mať žiaden parameter, ale nadeklarujeme v ňom pole hodnôt boolean, 100 hodnôt bude stačiť.

```
public KeyInput() {
keys = new boolean[100];
}
```

Vytvorenie metód triedy KeyInput. Okrem vopred pripravených tried z interfacu si vytvoríme metódu update, ktorá bude kontrolovať, či sa stlačila konkrétna klávesa. Ako prvú si pripravíme metódu keyPressed, môžeme si všimnúť, že ako parameter si berie KeyEvent key. A tak do pola pre kľúč key.getKeyCode() vložíme hodnotu pravda. To znamená, že pre kľúč rovný kódu znaku stlačenej klávesy sa jej hodnota zmení na hodnotu pravda, čiže je stlačená.

```
keys[key.getKeyCode()]=true;
```

No keď ju pustíme musí sa hodnota zmeniť na nepravdu, čiže rovnaký príkaz len s opačnou hodnotou zapíšeme aj do metódy keyReleased.

```
keys[key.getKeyCode()]=false;
```

Nakoniec v tejto triede upravíme metódu update, v nej naším premenným určujúcim smer vpravo a hore priradíme hodnoty z pola pre dané klávesy. Napríklad pre premennú vpravo priradíme hodnotu z pola pre kľúč KeyEvent.VK_D poprípade KeyEvent.VK_RIGHT, rovnako to aplikujeme aj pre premennú hore. To aké klávesy použijeme je osobná preferencia, môže to byť D a W alebo šípky na klávesnici ale aj čokoľvek iné čo nám vyhovuje. V tejto verzii sa budeme pohybovať len vpravo, poprípade vykonáme skok, ale hru si môže nakoniec každý upraviť ako sa mu bude páčiť, čiže môže doplniť aj pohyb smerom vľavo alebo prvok ako prikrčenie a podobne.

```
public void update() {
right = keys[KeyEvent.VK_D];
up = keys[KeyEvent.VK_W];
}
```

Vytvorenie inštancie triedy KeyInput. V triede Game si vytvoríme súkromnú premennú typu KeyInput, v konštruktore triedy Game následne vytvoríme novú inštanciu triedy KeyInput. A v metóde update spustíme metódu update triedy KeyInput.

```
private KeyInput key;
key = new KeyInput();
key.update();
```

Vytvoríme k nej getter.

```
public KeyInput getKey() {
return key;
}
```

Následne v triede Manager vytvoríme tiež getter, ktorý bude odkazovať na getter v triede Game.

```
public KeyInput getKey() {
return game.getKey;
}
```

Ďalej budeme upravovať triedy Player a Character. Najskôr vytvoríme novú premennú pre triedu Player, typu boolean, ktorá rozhoduje či je vpredu ľavá alebo pravá noha. K tomu ešte vytvoríme dve premenné typu long, jedna premenná bude uchovávať posledný čas kedy prebehla určitá časť kódu nazvime ju last a druhá premenná bude zaznamenávať koľko času prešlo medzi premennou last a súčasným časom nazvime ju delta.

```
private boolean step = true;
private long last, delta;
```

V konštruktore triedy Player, môžeme premennej delta priradiť nulu a premennej last priradíme súčasný čas v milisekundách.

```
delta=0;
last=System.currentTimeMillis();
```

V triede Character vytvoríme dve dedené premenné typu float, ktoré budú uchovávať vzdialenosť, ktorú máme prejsť v smere osi x a osi y, tak tieto premenné môžeme pomenovať xMove a yMove, môžeme ich deklaráciu pridať za rýchlosť, ktorú sme už deklarovali.

```
protected float speed, xMove, yMove;
```

Ďalej vytvoríme dedenú premennú typu boolean pomenujeme ju jump, premenná nám bude hovoriť či je náš hráč momentálne vo výskoku alebo nie.

```
protected boolean jump=false;
```

Nakoniec si vytvoríme pomocnú celočíselnú súkromnú premennú, ktorá bude zaznamenávať v akej pozícií výskoku sa hráč nachádza.

```
private int jumpPosition=0;
```

Premenným xMove a yMove priradíme v konštruktory hodnotu 0.

```
xMove = 0;
yMove = 0;
```

Ďalej vytvoríme tri metódy v triede Character. Metódu move, metódu moveX a metódu moveY. V metóde move len zavoláme metódy moveX a moveY. Metóda moveX bude navyšovať hodnotu x o hodnotu xMove. K metóde moveY sa vrátime za chvíľu.

```
public void move() {
moveX();
moveY();
}

public void moveX() {
x = x + xMove;
}

public void moveY() {

}
```

V triede Player vytvoríme metódu getInput. Metóda getInput: premenným xMove a yMove priradíme 0 a konečne sa dostaneme k podmienkam. Pokiaľ bolo stlačené tlačidlo „D“ alebo šípka vpravo, podľa toho ako to komu vyhovuje, tak premennej xMove priradíme premennú rýchlosť. V našom prípade teda xMove bude rovné 6.0f. Tu končí prvá podmienka, analogicky spravíme podmienku aj pre posun do hora. Okrem toho ešte vytvoríme podmienku pre zisťovanie, ktorá noha sa má vykresliť vpredu, táto časť sa deje len pokiaľ vykresľujeme hráča, ktorý má animáciu kroku. Je to jednoduché prepínanie medzi dvoma textúrami hráča. Pokiaľ majú študenti lepšieho umeleckého ducha a väčší zmysel pre detail, môže byť tento krok zložený z ľubovoľného počtu textúr, premenná step by však nemohla byť typu boolean ale typu int, pričom by rôzne Case vyjadrenia určovali, ktorý obrázok sa má vykresliť. Uvediem obe možnosti, ja budem používať jednoduchšiu a kratšiu variantu.

```
public void getInput() {
xMove = 0;
yMove = 0;
if(manager.getKey().right)
xMove = speed;
if(manager.getKey().up)
yMove = -speed;
delta = delta + (System.currentTimeMillis()-last);
last = System.currentTimeMillis();
if(delta>500) {
delta=0;
if(step) {
playerImage = Assets.playerBasic[0];
step=false;
}
else {
playerImage = Assets.playerBasic[1];
step=true;
}
}
```

```
}
```

A pre prípad viacerých prvkov animácie, uvedieme si príklad štyroch.

```
public void getInput() {
xMove = 0;
yMove = 0;
if(manager.getKey().right)
xMove = speed;
if(manager.getKey().up)
yMove = -speed;
delta = delta + (System.currentTimeMillis()-last);
last = System.currentTimeMillis();
if(delta>500) {
delta=0;
switch (step) {
case 0:
playerImage = Assets.playerBasic[0];
step = step + 1;
break;
case 1:
playerImage = Assets.playerBasic[1];
step = step + 1;
break;
case 2:
playerImage = Assets.playerBasic[2];
step = step + 1;
break;
case 3:
playerImage = Assets.playerBasic[3];
step = 0;
break;
}
}
}
```

V metóde update zavoláme metódu getInput a následne metódu move. Následne túto metódu musíme zavolať v triede Level a v jej metóde update. A túto metódu následne zavoláme v update triedy GameState.

```
@Override
public void update() {
getInput();
move();
}
```

Dokončíme metódu moveY v triede Character, pomocou boolean premennej jump a int premennej jumpPosition vytvoríme ďalšie podmienky, ktoré zistia či má hráč stále stúpať alebo sa nachádza v bode kedy začne klesať.

```
public void moveY() {
if(yMove!=0 && !jump) {
y = y + yMove;
jump = true;
}
if(jump && jumpPosition<15) {
y = y - speed;
jumpPosition ++;
}
if(jump && jumpPosition>14) {
y = y + speed;
jumpPosition ++;
}
if(jump && jumpPosition==31) {
jump=false;
jumpPosition=0;
}
}
```

Pre úplný chod programu je ešte nutné v triede Game v metóde init priradiť nášmu JFramu KeyListener.

```
gameScene.getFrame().addKeyListener(key);
```

8.5.3 Výsledok cvičenia:

Študenti vytvoril novú triedu implementujúcu interface KeyListener, pomocou tejto triedy sa pripravilo jednoduché ovládanie hráča. S týmto nastala situácia keď hráč môže opustiť vykreslenú obrazovku, čo nie je vhodné. Študenti tiež mali príležitosť vyskúšať si tvorbu podmienok if a prácu s polom.

8.6 Cvičenie 5:

8.6.1 Predpoklady k cvičeniu:

cyklus

8.6.2 Úloha:

V tomto cvičení sa zameriame na odstránenie problému keď hráč odíde z obrazovky. Do nášho projektu pridáme novú triedu Camera, ktorej úlohou bude sledovať hráča počas jeho pohybu po levely.

Vytvorenie novej triedy Camera. Túto triedu vytvoríme v balíku e_graphics. V našej hre sa hráč posúva len po osi x, respektíve po osi y len vykonáva skok a teda nemôže opustiť rámec obrazovky. Z toho dôvodu vytvoríme súkromnú premennú len pre súradnice x-ovej osi, plus inštanciu triedy Manager.

```
private float xPos;
private Manager manager;
```

Vytvorenie konštruktora triedy Camera. Konštruktor bude mať dva vstupné parametre jedným bude inštancia triedy Manager a druhým premenná typu float.
V konštruktory následne nainicializujeme naše premenné.

```
public Camera(Manager manager, float xPos) {
this.manager = manager;
this.xPos = xPos;
}
```

Vytvorenie metód pre triedu Camera. Metódy, ktoré budeme potrebovať sú getter a setter na premennú xPos, metóda move a metóda focusOnObject.

```
public float getxPos() {
return xPos;
}

public void setxPos(float xPos) {
this.xPos = xPos;
}
```

Začneme s metódou focusOnObject jej vstupný parameter bude prekvapivo inštancia triedy Object. A metóda spôsobí jedine, že premennej xPos priradí istú hodnotu na základe pozície objektu.

```
public void focusOnObject(Object object) {
xPos = object.getX() - manager.getWidth() / 2;
}
```

Najskôr vezme pozíciu hráča a následne od nej odpočíta polovicu šírky nášho herného plátna. To spôsobí, že pozícia hráča teda bude vždy o polovicu plátna pred pozíciou kamery.

Metóda move len posunie pozíciu kamery o hodnotu, ktorú metóda dostala ako parameter.

```
public void move(float xVal) {
xPos = xPos + xVal;
}
```

To je v triede Camera všetko. Teraz si triedu nainplementujeme v triede Game. Najskôr súkromná premenná, potom inicializácia v metóde init.

```
private Camera camera;
camera = new Camera(manager, 0);
```

O distribuovanie inštancie sa opäť bude starať trieda Manager. Tak v nej vytvoríme súkromnú premennú a tiež k nej vytvoríme getter a setter.

```
private Camera camera;

public Camera getCamera() {
return camera;
}

public void setCamera(Camera camera) {
this.camera = camera;
}
```

Teraz môžeme zavolať metódu setCamera v triede Game, čím bude trieda Manager schopná voľne distribuovať inštanciu triedy Camera.

```
manager.setCamera(camera);
```

Následne zavoláme metódu focusOnObject, v triede Player v metóde update. Tým sa naša kamera bude stále zameriavať len na hráča.

```
manager.getCamera().focusOnObject(this);
```

Nakoniec ešte zmeníme vykresľovanie hráča v metóde render. Miesto čisto pozície x, ktorú dostane pri svojej inicializácií sa vykreslí na pozícií x – pozícia kamery, aby bol hráč stále vykresľovaný v strede plátna.

```
graphics.drawImage(playerImage, (int) ((x - manager.getCamera().getxPos())), (int) y,
null);
```

Pre našu hru však vykresľovanie hráča úplne na stred nie je vhodné a tak ho vykreslíme o trochu viac vľavo. To dosiahneme ľahko len jeho x-ovú súradnicu pri vykresľovaní vydelíme štvorkou.

```
graphics.drawImage(playerImage, (int) ((x - manager.getCamera().getxPos())/4), (int) y,
null);
```

Po spustení si ale všimneme, že hráč sa neposúva vpravo keď mu zadáme klávesový príkaz. To je však len optický klam pretože ako náhle sa hráč posunie, posunie sa s ním aj kamera a teda to vyzerá, že sa hráč neposunul. Aby sme zachytili jeho posun môžeme si spraviť pohyblivé pozadie.

V triede Level si vytvoríme inštanciu triedy Player a dve premenné typu int. Premenná bg nám bude poskytovať informácie o momentálnej súradnici x, na ktorej sa vykresľuje pozadie. Premenná last nám bude ukladať údaj o poslednej x-ovej pozícií hráča.

```
private Player player;
private int bg=0, last;
```

Premennej last môžeme v konštruktore priradiť aktuálnu pozíciu hráča.

```
last = (int) player.getX();
```

Upravíme metódu update, po tom čo sa spustí metóda update v triede hráč upravíme premenné bg a last tak aby odpovedali pozícií, ktorej majú odpovedať.

```
bg = bg - (int) (player.getX() - last);
last = (int) player.getX();
```

Od premennej bg odpočítavame, pretože naše pozadie sa má pohybovať v smere vľavo, keď sa náš hráč hýbe vpravo. Tu však dostávame problém, kde sa pozadie prestalo vykresľovať v pravej časti plátna. Tento problém vyriešime tým, že budeme pozadie vykresľovať dva krát vedľa seba v metóde render.

```
graphics.drawImage(Assets.background, bg, 0, null);
graphics.drawImage(Assets.background, bg+manager.getWidth(), 0, null);
```

Následne však nastane rovnaký problém po tom čo prejdeme obe pozadia. Do metódy update teda ešte pridáme podmienku, ktorá pokiaľ hodnota premennej bg bude taká, že sa už nevykreslí na plátno tak premennú bg vynulujeme.

```
if(bg<=manager.getWidth()*-1) {
bg=0;
}
```

Teraz už dokážeme zaznamenať, že hráč sa pohybuje aj keď ho kamera stále vykresľuje na stred obrazovky.

Pohybujeme sa však vo vzduchu čo nie je moc prirodzené a tak si vykreslíme podlahu, po ktorej budeme kráčať.

Bude sa jednať len o veľmi primitívnu podlahu, ktorú vytvoríme v cykle for. Bude tvorená dielcami, ktoré budú mať rovnakú výšku a šírku. Na to si vytvoríme dve súkromné stále celočíselné premenné floorWidth a floorHeight budú mať veľkosť 100 a 30.

```
private final static int floorHeight = 30, floorWidth = 100;
```

Teraz si môžeme vytvoriť cyklus, v ktorom sa dielce vykreslia. Bude spustený v metóde render a bude sa v našom prípade opakovať 10 krát. Aby sme sa ale vyhli tomu, že mu na pevno určíme desať opakovaní bude cyklus prebiehať (šírka hry/šírka dielcu) krát. U nás to teda bude 1000/100 čo je 10 no keď zmeníme niektorú hodnotu či už šírku hry alebo šírku dielca, cyklus prebehne toľko krát aby bola podlaha vykreslená správne.

```
for(int i=0; i<manager.getWidth()/floorWidth; i++) {
graphics.drawImage(Assets.stone, i*floorWidth, manager.getHeight()-floorHeight, floor-
Width, floorHeight, null);
}
```

Môžeme si všimnúť, že tento krát sme pri vykresľovaní použili viac parametrov. Okrem parametru čo sa má vykresliť, súradníc kde sa to má vykresliť a observru sme zadali aj šírku a výšku vykreslenia.

Nakoniec cvičenia ešte vykreslíme hráča na úroveň podlahy. Po tom čo ho nainicializujeme v konštruktory triedy Level, zavoláme metódu setY, ktorá určí y-ovú súradnicu nášho hráča.

```
player.setY(manager.getHeight()-player.getHeight()-floorHeight);
```

8.6.3 Výsledok cvičenia:

Študenti vytvorili novú triedu Camera, ktorej účelom je vždy vykresliť hráča na rovnakú pozíciu obrazovky. Tiež sme vytvorili pohyblivé pozadie, ktoré dodáva pocit, že náš hráč sa naozaj pohybuje po levely a vytvorili sme podlahu levelu pričom sme prvý krát použili cyklus.

8.7 Cvičenie 6:

8.7.1 Predpoklady k cvičeniu:

úvod do zoznamov

8.7.2 Úloha:

Cvičenie je zamerané na ďalšiu prácu s objektami. Vytvoríme novú triedu Monster. Trieda Monster bude rozširovať triedu Character.

Vytvorenie triedy Monster, triedu vytvoríme v balíku e_objects a bude rozširovať triedu Character.

```
public class Monster extends Character{
```

Pred prácou v tejto triede si ešte vytvoríme verejnú stálu premennú určujúcu základnú šírku monštra. Hodnota tejto premennej bude 100.

```
public static final int defWidth = 75, defHeight = 100, defMonsterWidth = 100;
```

Vytvorenie konštruktora triedy Monster, rovnako ako v prípade konštruktora triedy Player vynecháme parametre pre šírku a výšku. Keďže potrebujeme v odkazovaní na supertriedu použiť údaje o šírke a výške použijeme základné údaje z triedy Character. Základnú šírku monštra a základnú výšku, pokiaľ by študenti používali vlastné textúry, ktoré by boli rôznych rozmerov museli by si pripraviť aj premennú základnej výšky monštra, v mojom prípade sa jedná o rovnakú výšku akú má postava hráča a tak môžeme použiť základnú výšku.

```
public Monster(Manager manager, float x, float y) {
```

```
super(manager, x, y, defMonsterWidth, defHeight);
```

Vytvorenie premenných v triede Monster. Vytvoríme dve súkromné premenné, jedna z nich, premenná monsterType, bude pomocná celočíselná premenná, ktorá bude odkazovať na typ monštra, ktoré sa bude vykresľovať. Druhá premenná je typu BufferedImage a predstavuje textúru, ktorá sa vykreslí na základe premennej monsterType, spomenutej vyššie.

```
private BufferedImage monsterImage;
private int monsterType = 0;
```

V konštruktore následne vytvoríme sled príkazov, ktorý pri vytvorení monštra náhodne určí o aký typ sa bude jednať, podľa tohto typu sa premennej monsterImage priradí odpovedajúcu textúru tohto typu.

```
Random rand = new Random();
monsterType = rand.nextInt(3) + 1;
switch(monsterType){
case 1: monsterImage = Assets.waterMonster;
break;
case 2: monsterImage = Assets.grassMonster;
break;
case 3: monsterImage = Assets.fireMonster;
break;
default: monsterImage = Assets.waterMonster;
}
}
```

Metódy v triede Monster, okrem dedených metód update a render si vytvoríme metódu reset, ktorej účel bude po odstránení monštra vytvoriť nové monštrum a metódu getMonsterType.

Metóda update. V tejto metóde si nadefinujeme rýchlosť pohybu monštra a priebežne budeme informovať o zmene zdravia monštra. Zmenu zdravia si môžu študenti v tomto cvičení vyskúšať pridaním odpočtu 1 pri volaní metódy setHealthBar. Je tiež možné vyskúšať si zmenu rýchlosti, to dokážeme úpravou premennej xMove.

```
@Override
public void update() {
this.xMove = -5;
this.setHealthBar(this.getHealthBar());
this.moveX();
}
```

Metóda render. V tejto metóde prebehne jedine vykreslenie monštra. Ako textúra sa použije premenná monsterImage, súradnicu na x-ovej osi zmeníme tak aby bolo monštrum vykreslené mimo obrazovky a teda odpočítame hodnotu manager.getCamera().getxPos(), celú túto hodnotu ešte vynásobíme dvojkou. Ako súradnicu y použijeme parameter y z konštruktora.

```
@Override
public void render(Graphics graphics) {
graphics.drawImage(monsterImage, (int) ((x - manager.getCamera().getxPos())*2), (int)
y, null);
}
```

Metóda reset. V metóde reset prebehne rovnaký postup ako v konštruktore triedy, keďže sa však jedná o nové monštrum je nutné mu priradiť opäť plné zdravie a novú štartovnú pozíciu.

```
public void reset() {
Random rand = new Random();
monsterType = rand.nextInt(3) + 1;
switch(monsterType) {
case 1: monsterImage = Assets.waterMonster;
break;
case 2: monsterImage = Assets.grassMonster;
break;
case 3: monsterImage = Assets.fireMonster;
break;
default: monsterImage = Assets.waterMonster;
}
this.setHealthBar(defHealth);
this.x=100;
}
```

Metóda getMonsterType. Jedná sa o obyčajný getter premennej monsterType.

```
public int getMonsterType() {
return monsterType;
}
```

To je celá úprava triedy Monster, následne ešte upravíme triedu Level.

V triede Level si nainicializujeme inštanciu triedy Monster.

```
private Monster monster;
monster = new Monster(manager, 100, 100);
```

Následne výšku, v ktorej sa monštrum vykresľuje aby odpovedala výške podlahy.

```
monster.setY(manager.getHeight()-monster.getHeight()-floorHeight);
```

V metóde render následne pridáme vykresľovanie monštra. To nastane v prípade, že jeho zdravie je nad 0.

```
if(monster.getHealthBar() > 0)
monster.render(graphics);
```

Úvodná práca so zoznamom prebehne tiež v triede Level. Vytvoríme si nový zoznam.

```
ArrayList<Integer> killedMonsters = new ArrayList<Integer>();
```

Do tohto zoznamu budeme pridávať údaje o type monštra, ktoré zomrelo. To prebehne v metóde update triedy Level, vytvoríme podmienku, ktorá v jednej vetve spúšťa metódu render triedy Monster, ak však zdravie monštra dosiahne nulu, spustí sa druhá vetva s metódou reset triedy Monster.

```
if(monster.getHealthBar() == 0) {
killedMonsters.add(monster.getMonsterType());
monster.reset();
}
else
monster.update();
```

Pre vyskúšanie je možné do metódy update pustiť príkaz na výpis tohto zoznamu.

```
System.out.print(killedMonsters);
```

Keďže sme zatiaľ nenaprogramovali útok hráča, smrť monštra môže byť spôsobené jedine tým, že budeme zdravie odpočítavať v metóde update triedy Monster.

8.7.3 Výsledok cvičenia:

Študenti vytvoril novú triedu Monster, ktorá bude predstavovať nepriateľov v hre. Následne sa upravila trieda Level tak aby týchto nepriateľov bolo možné vykresliť. Tiež si vyskúšali prvú prácu so zoznamom.

8.8 Cvičenie 7:

8.8.1 Predpoklady k cvičeniu:

enum

8.8.2 Úloha:

V tomto cvičení sa vytvorí nová trieda Attack, predstavujúca útok hráča. Pridá sa tiež nové ovládanie na inicializáciu útoku a na zmenu postavičky hráča.

Vytvorenie triedy Attack, trieda bude vytvorená v balíku e_objects a bude rozširovať triedu Object.

```
public class Attack extends Object{
```

Vytvorenie konštruktora triedy Attack. Už klasicky vynecháme parametre šírky a výšky, v prípade tejto triedy použijeme ako údaje súkromnú stálu premennú attackSize, ktorú si vytvoríme v triede Object a bude mať hodnotu 40.

```
public static final int attackSize = 40;
```

Konštruktor teda vyzerá nasledovne.

```
public Attack(Manager manager, float x, float y) {
super(manager, x, y, attackSize, attackSize);
```

Vytvorenie premenných triedy Attack, trieda bude mať premennú predstavujúcu textúru útoku typu BufferedImage a pomocné premenné pre súradnice x a y typu float.

```
private float posX, posY;
private BufferedImage attackImage;
```

K premenných posX a posY vytvoríme gettre a settre.

```
public float getPosX() {
return posX;
}

public void setPosX(float posX) {
this.posX = posX;
}

public float getPosY() {
return posY;
}

public void setPosY(float posY) {
this.posY = posY;
}
```

Teraz si vytvoríme vymenovací typ, enum. Ten vytvoríme v triede Character.

```
enum Type {
water,
grass,
fire,
normal
}
```

Pokračujeme v práci v triede Character, vytvoríme si dve nové premenné.

```
private Type characterType;
private boolean attack;
```

K týmto premenným si vytvoríme gettre a settre.

```
public void setType(Type characterType) {
this.characterType = characterType;
}

public Type getType() {
return this.characterType;
}

public void setAttacked(boolean attack) {
this.attack = attack;
}

public boolean getAttacked() {
return this.attack;
}
```

Pridanie nových akcií pri stlačení klávesy, pripravíme v triede KeyInput. Najskôr pridáme nové pomocné premenné, celočíselnú premennú, ktorá bude predstavovať typ postavy, na ktorú sa hráč chce zmeniť a boolean premennú attack.

```
public int type;
public boolean attack;
```

Následne v metóde update pridáme akcie, ktoré pri stlačení danej klávesy nastanú.

```
attack = keys[KeyEvent.VK_A];
if(keys[KeyEvent.VK_0])
type = 0;
else if(keys[KeyEvent.VK_1])
type = 1;
else if(keys[KeyEvent.VK_2])
type = 2;
else if(keys[KeyEvent.VK_3])
type = 3;
```

Úprava triedy Monster. Pridanie nových premenných, úprava konštruktora a úprava metód update a render. Najskôr nové premenné.

```
private static int attackSpeed = 10;
Attack attack;
```

Konštruktor je upravený aby na základe typu monštra priradil odpovedajúci útok rovnakého typu pre monštrum.

```
public Monster(Manager manager, float x, float y) {
super(manager, x, y, defMonsterWidth, defHeight);
Random rand = new Random();
monsterType = rand.nextInt(3) + 1;
switch(monsterType)  {
case 1: monsterImage = Assets.waterMonster;
this.setType(Type.water);
break;
case 2: monsterImage = Assets.grassMonster;
this.setType(Type.grass);
break;
case 3: monsterImage = Assets.fireMonster;
this.setType(Type.fire);
break;
default: monsterImage = Assets.waterMonster;
this.setType(Type.water);
}
}
```

Upravená metóda update, v metóde je pridaná podmienka zaručujúca, že na hráča letí len jeden útok, zároveň zabezpečuje, že útoky nie sú periodické ale náhodné.

```
public void update() {
this.xMove = -2;
move();
this.setHealthBar(this.getHealthBar());
Random rand = new Random();
if(rand.nextInt(50) == 1 && !this.getAttacked()) {
attack = new Attack(manager, this.getType(), (int) ((x - mana-
ger.getCamera().getxPos())) * 2, manager.getHeight() - 110);
this.setAttacked(true);
}

if(this.getAttacked()) {
if(attack.getPosX() - attackSpeed <= manager.getCamera().getxPos()*2) {
this.setAttacked(false);
attack = null;
}
else
attack.setPosX((attack.getPosX() - attackSpeed));
}
}
```

V upravenej metóde render sa pridáva vykresľovanie útoku monštra.

```
public void render(Graphics graphics) {
graphics.drawImage(monsterImage, (int) ((x - manager.getCamera().getxPos())) * 2 ,
(int) y, null);
if(this.getAttacked()) {
attack.x -= attackSpeed;
attack.render(graphics);
}
```

```
}
```

Úprava triedy Player. Vytvoríme dve nové premenné, inštančnú premennú triedy Attack a súkromnú stálu celočíselnú premennú predstavujúcu rýchlosť útoku.

```
private static int attackSpeed = 10;
Attack attack;
```

Update a render metódy v triede Player prejdú podobnou zmenou ako v triede Monster, metóda getInput prejde väčšou zmenou kde pridávame útok a štyri varianty zmeny typu postavy.

```
public void update() {
getInput();
move();
if(this.getAttacked()) {
if(attack.getPosX() + attackSpeed >= manager.getWidth()) {
this.setAttacked(false);
attack = null;
}
else
attack.setPosX(attack.getPosX() + attackSpeed);
}
manager.getCamera().focusOnObject(this);
}

public void render(Graphics graphics) {
graphics.drawImage(playerImage, (int) ((x - manager.getCamera().getxPos())/4), (int) y,
null);
if(this.getAttacked()) {
attack.x += attackSpeed;
attack.render(graphics);
}
}

public void getInput() {
xMove = 0;
yMove = 0;
switch(manager.getKey().type)     {
case 1: this.setType(Type.water);
playerImage = Assets.playerWater[0];
break;
case 2: this.setType(Type.grass);
playerImage = Assets.playerGrass[0];
break;
case 3: this.setType(Type.fire);
playerImage = Assets.playerFire[0];
break;
case 0: this.setType(Type.normal);
playerImage = Assets.playerBasic[0];
break;
```

```
}

if(manager.getKey().attack && !this.getAttacked()) {
attack = new Attack(manager, this.getType(), ((x - manager.getCamera().getxPos())/4) +
10, manager.getHeight() -400);
attack.setY(manager.getHeight()-this.getHeight()-30);
this.setAttacked(true);
}

if(manager.getKey().right) {
xMove = speed;
}
if(manager.getKey().up) {
yMove = -speed;
}
delta = delta + (System.currentTimeMillis()-last);
last = System.currentTimeMillis();
if(delta>500) {
delta=0;
if(step) {
if(this.getType() == Type.normal)
playerImage = Assets.playerBasic[0];
else if(this.getType() == Type.water)
playerImage = Assets.playerWater[0];
if(this.getType() == Type.grass)
playerImage = Assets.playerGrass[0];
if(this.getType() == Type.fire)
playerImage = Assets.playerFire[0];
step=false;
}
else {
if(this.getType() == Type.normal)
playerImage = Assets.playerBasic[1];
else if(this.getType() == Type.water)
playerImage = Assets.playerWater[1];
if(this.getType() == Type.grass)
playerImage = Assets.playerGrass[1];
if(this.getType() == Type.fire)
playerImage = Assets.playerFire[1];
step=true;
}
}
}
```

Následne upravíme konštruktor triedy Attack. Pridáme podmienky, ktoré nám do premennej pridajú správnu textúru na základe typu postavy.

```
public Attack(Manager manager, Type charType, float x, float y) {
super(manager, x, y, attackSize, attackSize);
if(charType == Type.water)
attackImage = Assets.water;
```

```
if(charType == Type.grass)
attackImage = Assets.grass;
if(charType == Type.fire)
attackImage = Assets.fire;
if(charType == Type.normal)
attackImage = Assets.basic;
}
```

Pri úprave metód triedy Attack, vykreslíme textúru útoku na obrazovku v metóde render a pridáme metódu pre výpočet hodnoty útoku.

```
public void render(Graphics graphics) {
graphics.drawImage(attackImage, (int) this.getX(), (int) this.getY(), null);
}

public int attack(Type monsterType, Type playerType) {
if(monsterType == Type.water && playerType == Type.grass
|| monsterType == Type.grass && playerType == Type.water)
return 3;
else if(monsterType == Type.fire && playerType == Type.grass
|| monsterType == Type.grass && playerType == Type.fire)
return 1;
else
return 2;
}
```

Nakoniec ešte upravíme metódu update v triede Level.

```
public void update() {
player.update();
if(monster == null){
Random rand = new Random();
if(rand.nextInt(50) == 1)
monster.reset();
}
else if(monster.getHealthBar() == 0) {
monster = null;
killedMonsters.add(monster.getMonsterType());
}
else
monster.update();
bg = bg - (int) (player.getX() - last);
last = (int) player.getX();
if(bg<=manager.getWidth()*-1) {
bg=0;
}
}
```

8.8.3 Výsledok cvičenia:

Študenti si vyskúšali prácu s typom enum. Do projektu pridali schopnosť útočiť ako u hráča tak u monštra, hráč má navyše možnosť zmeniť svoj typ postavy na základe stlačenia príslušnej klávesy.

8.9 Cvičenie 8:

8.9.1 Predpoklady k cvičeniu:

8.9.2 Úloha:

V tomto cvičení sa dokončí práca na projekte, samozrejme je možné projekt ďalej rozširovať podľa vlastného uváženia. Od študentov sa očakáva, že doladia detaily, ktoré v projekte zatiaľ chýbajú. Kolízie, počítanie skóre a podobne. Študenti využívajú poznatky, ktoré nadobudli počas predchádzajúcich cvičení a prednášok v rámci kurzu. Študenti majú priestor ďalej túto hru vylepšovať pokiaľ ich tento návrh zaujal. Ako ďalšie vylepšenia sa dajú uviesť: úprava plynulosti hry, zmenenie náročnosti hry pokiaľ hráč dosiahne určité skóre, napríklad zvýšenie rýchlosti monštra. Okrem týchto kozmetických doplnkov návrh poskytuje možnosti rozšírenia pre ďalšie prvky programovania, ktoré v tejto práci neboli použité ako napríklad použitie databázy, kde by študenti boli schopní svoje skóre uložiť do databázy, ktorá by bola pripravená alebo by si ju pripravili sami.

Záver

V praktickej časti tejto práce som navrhol postupne vylepšovaný program, ktorý by sa dal použiť vo výuke vstupného kurzu programovania. Tento program pokrýva základné prvky objektovo orientovaného programovania, ktoré by študenti mali na konci kurzu ovládať. Prerekvizitou k dokončeniu cvičení sú základné znalosti o danej téme, ktoré je nutné študentom poskytnúť pred konkrétnym cvičením. Cvičenia boli písané formou návodu, ktorý môže vyučujúci poskytnúť študentom. Jednotlivé cvičenia programu boli navrhnuté tak aby boli zvládnuteľné počas jedného cvičenia, tento aspekt však nebol objektívne otestovaný.

V rámci teoretickej časti som poukázal na momentálny stav výuky programovania. Vytvoril som krátky prehľad vhodných programovacích jazykov a vývojových prostredí, ktoré pokladám za vhodné pre výuku programovania vo vstupných kurzoch. Okrem toho som tiež uviedol ďalšie možné návrhy pre postupne vylepšovaný program v žánre hier.

Zhodnotil som taktiež základné výhody a nevýhody postupne vylepšovaného programu, tento bod by bolo vhodné podporiť konkrétnymi príkladmi takéhoto typu programu použitého vo výuke. V tomto bode som sa stretol s komplikáciou kde výuka cvičení záleží na samotnom vyučujúcom, ktorý volí najlepší spôsob výuky podľa vlastného odhadu, toho čo je pre študenta najvhodnejšie a najprínosnejšie. Okrem toho kurzy, nie len vysokoškolské, ale aj rôznych technických komunít, ktoré poskytujú výuku programovania na profesionálnejšej úrovni, nezvyknú zverejňovať svoje postupy keďže by to bolo nežiadúce pre ich činnosť. Tieto body spôsobili, že som nenašiel veľa príkladov použitia postupne vylepšovaných programov vo vstupných kurzoch programovania.

Použité informačné zdroje

[1] NOVÁK, Marek, BIŇAS, Miroslav, MICHALKO, Miroslav, JAKAB, František. Ako motivovať študentov softverového inžinierstva, Technická univerzita Košice, [online] [cit. 2019-03-20], Adresa: <https://www.researchgate.net/profile/Marek_Novak/publication/228458474_Ako_motivovat_studentov_softveroveho_inzinierstva/links/00b4951c2db5a1114e000000/Ako-motivovat-studentov-softveroveho-inzinierstva.pdf>

[2] PECINOVSKÝ, Rudolf. Výuka programování pro praxi, Vysoká škola ekonomická v Praze, Fakulta informačních technologií, [online] [cit. 2019-03-23], Adresa: <http://vyuka.pecinovsky.cz/prispevky/2008_IN_Vyuka_PGM_pro_praxi.pdf>

[3] MILLER, Bradley N., RANUM, David L. Problem Solving with Algorithms and Data Structures using Python, 2. vydanie. Franklin Beedle & Associates, 2005, ISBN 1590282574

[4] LEDNÁR, Matej. Príručka programátora – Prehľadný sprievodca jazykom JavaScript 1.5+. Bratislava: MLD Group, 2009, ISBN 978-80-89448-02-9

[5] WALLACE, Charles. The Semantics of the C++ Programming Language, 1993-11-23, [online] [cit. 2018-12-10], Adresa: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.35.6692&rep=rep1&type=pdf>

[6] FERGUSON, Andrew. A History of Computer Programming Languages, [online] [cit. 2018-11-05], Adresa: <http://cs.brown.edu/~adf/programming_languages.html>

[7] Oracle, [online] [cit. 2018-11-06], Adresa:<https://www.oracle.com/index.html>

[8] Java, [online] [cit. 2018-11-06], Adresa: <https://java.com/>

[9] CLAYBERG, Eric, RUBEL, Dan. Eclipse plug-ins (3rd Edition). Boston: Pearson Education, Inc., 2009. ISBN 0321553462.

[10] JetBrains, [online] [cit. 2019-03-15], Adresa: <https://www.jetbrains.com/idea/>

[11] Minecraft, [online] [cit. 2018-11-11] Adresa: <https://minecraft.net/>

[12] Unreal Engine, [online] [cit. 2018-11-11] Adresa:

<https://www.unrealengine.com/en-US/what-is-unreal-engine-4>

[13] Unity, [online] [cit. 2018-11-11] Adresa: <https://unity3d.com/>

[14] GALČÍK, František, GURSKÝ, Peter, NOVOTNÝ, Róbert. Úvodný kurz programovania v Jave na PF UPJŠ, [online], [cit. 2019-03-17], Adresa:

<http://ics.upjs.sk/~gursky/papers/2011didinfo.pdf>

[15] PECINOVSKÝ, Rudolf. OOP - Learn Object Oriented Thinking & Programming, Řepín-Živonín, Tomáš Bruckner, 2013, ISBN 978-80-904661-9-7

Obrázky:
[1] Diagram znázorňujúci dizajn správania zvieraťa podľa schopnosti lietať a vydávať zvuk, Adresa:
<https://en.wikipedia.org/wiki/Composition_over_inheritance#/media/File:UML_diagram_of_composition_over_inheritance.svg>