Univerzita Pardubice
Fakulta elektrotechniky a informatiky

OOP4FUN: OBJECT ORIENTED PROGRAMMING FOR FUN

OOP4FUN: OBJECT ORIENTED PROGRAMMING FOR FUN – Příloha k průvodci pro současné učitele středních škol k výuce programování: Učební osnova pro výuku programování podle principů „light oop“

Editoři

Michal Varga, Josef Rak, Dušan Savić, Zlatko Stapić

Autoři:

Peter Sedláček, Nika Kvaššayová, Jozef Kostolný, Michal Mrena, Patrik Rusnák, Peter Sobe, Ilija Antović, Miloš Milić, Tatjana Stojanović, Davor Fodrek, Lidija Kozina, Marko Mijač, Dijana Plantak Vukovac, Antonela Čižmešija, Dijana Peras, Goran Hajdin, Lea Masnec

Pardubice, září 2024.

Co-funded by the
Erasmus+ Programme
of the European Union

Projekt	Object Oriented Programming for Fun
Zkratka projektu	OOP4FUN
Číslo projektu	2021-1-SK01-KA220-SCH-00027903
Hlavní řešitel	Žilinská univerzita v Žiline (Slovakia)
Řešitelé projektu	Sveučilište u Zagrebu (Croatia)
	Srednja škola Ivanec (Croatia)
	Univerzita Pardubice (Czech Republic)
	Gymnázium, Pardubice, Dašická 1083 (Czech Republic)
	Obchodna akademia Povazska Bystrica (Slovakia)
	Hochschule fuer Technik und Wirtschaft Dresden (Germany)
	Gymnasium Dresden-Plauen (Germany)
	Univerzitet u Beogradu (Serbia)
	Gimnazija Ivanjica (Serbia)
Rok publikace	2024

Prohlášení o vyloučení odpovědnosti
Financováno Evropskou unií. Názory vyjádřené jsou názory autora a neodráží nutně oficiální stanovisko Evropské unie či Slovenské akademické asociace pro mezinárodní spolupráci (SAAIC). Evropská unie ani SAAIC za vyjádřené názory nenese odpovědnost.

Obsah

1. Informační list ... 5
1.1. Popis předmětu ... 5
1.2. Charakteristika předmětu ... 5
1.3. Cíl předmětu ... 5
1.4. Výsledky vzdělávání ... 5
1.5. Prostorové a technické požadavky ... 5
2. Zásady učební osnovy ... 7
3. Projekty ... 11
3.1. Bomberman ... 12
3.1.1. Obsah a rozsah vzdělávacího programu ... 12
3.1.2. Rozdělení projektu ... 13
3.2. Tower defense ... 35
3.2.1. Obsah a rozsah vzdělávacího programu ... 35
3.2.2. Rozdělení projektu ... 36
3.3. Projekt Mravenci ... 54
3.3.1. Rozdělení projektu ... 54
4. Seznam použité literatury ... 66
5. Přílohy ... 66
5.1. Export návrhu vzdělávání pro projekt Bomberman ... 66
5.2. Export návrhu vzdělávaní pro projekt Tower defense ... 66
5.3. Export návrhu vzdělávání pro projekt Mravenci ... 66

Seznam obrázků

Obrázek 1: Prostředí Greenfoot s konečným stavem projektu Bomberman ... 12
Obrázek 2: Pracovní zátěž studenta při řešení projektu Bomberman ... 13
Obrázek 3: Prostředí Greenfoot s konečným stavem projektu Tower defense ... 35
Obrázek 4: Pracovní zátěž studenta při řešení projektu Tower defense ... 36
Obrázek 5: Konfigurace vlastních nastavení instancí na předpovídání pohybu instance třídy Enemy . 41
Obrázek 6: Konfigurace specifické nastavení instancí na předpovídání pohybu instance třídy Enemy 41
Obrázek 7: UML sekvenční diagram interakce instance třídy Enemy s jinými objekty při dosáhnutí instance třídy Orb ... 45
Obrázek 8: UML sekvenční diagram interakce instance třídy Tower s jinými objekty při vytváření instancí třídy Bullet ... 47
Obrázek 9: UML sekvenční diagram interakce instance třídy Bullet s ostatními objekty při zásahu instance třídy Enemy ... 48

Seznam tabulek

Tabulka 1: Konstruktivní směřování projektu Bomberman 13
Tabulka 2: Konstruktivní směrováni projektu Tower defense 36
Tabulka 3: Porovnaní pracovní zátěže kapitoly úvod do prostředí Greenfoot mezi projekty Bomberman a Tower defense 37
Tabulka 4: Porovnaní pracovní zátěže kapitoly definice tříd mezi projekty Bomberman a Tower defense 38
Tabulka 5: Porovnaní pracovní zátěže kapitoly algoritmizace mezi projekty Bomberman a Tower defense 39
Tabulka 6: Porovnaní pracovní zátěže kapitoly větvení mezi projekty Bomberman a Tower defense 40
Tabulka 7: Porovnaní pracovní zátěže kapitoly proměnné a výrazy mezi projekty Bomberman a Tower defense 42
Tabulka 8: Porovnaní pracovní zátěže kapitoly asociace mezi projekty Bomberman a Tower defense 44
Tabulka 9: Porovnaní pracovní zátěže kapitoly dědičnost mezi projekty Bomberman a Tower defense 49
Tabulka 10: Porovnání pracovní zátěže kapitoly zapouzdření mezi projekty Bomberman a Tower defense 52
Tabulka 11: Porovnaní pracovní zátěže kapitoly úvod do prostředí Greenfoot mezi projekty Bomberman a Mravenci 55
Tabulka 12: Porovnaní pracovní zátěže kapitoly definice třídy a základní práce se třídami mezi projekty Bomberman a Mravenci 55
Tabulka 13: Porovnaní pracovní zátěže kapitoly zapouzdření, kompozice, metody v projektu Mravenci a podobné kapitoly v projektu Bomberman - zapouzdření 57
Tabulka 14: Porovnaní pracovní zátěže kapitol konstruktory, složitější volání metod (práce s grafikou v prostředí Greenfoot) v projektu Mravenci a podobné kapitoly v projektu Bomberman - algoritmus, ovládací prvky aplikace, vytváření metod 58
Tabulka 15 Porovnání pracovní zátěže kapitoly větvení, podmíněné vykonávání mezi projekty Bomberman a Mravenci 59
Tabulka 16: Porovnaní pracovní zátěže algoritmus, výčtový typ (enum), pole v projektu Mravenci a podobné kapitoly v projektu Bomberman - seznam a for each cyklus 60
Tabulka 17: Porovnaní pracovní zátěže kapitoly zpracování uživatelského vstupu, logika hry v projektu Mravenci a podobné kapitoly v projektu Bomberman - algoritmus, ovládací prvky aplikace, vytváření metod 63

1. Informační list

1.1. Popis předmětu

Cílem předmětu je naučit studenty řešit úlohy z programování s využitím základů objektově orientovaného programování (OOP) podle myšlenky „zjednodušeného“ OOP. Studenti se naučí rozdělit zadané úlohy mezi spolupracující objekty, určit jejich kompetence a implementovat navržený model. Předmět nevyžaduje žádné předchozí zkušenosti a znalosti programování. Vyučuje se v programovacím jazyce Java. Předmět vysvětluje „zjednodušené“ koncepty OOP (jako je zapouzdření, dědičnost nebo asociace) na tvorbě počítačových her, kde se tyto koncepty využívají intuitivním způsobem. Proces tvorby počítačové hry je založený na týmové práci a prakticky využívá vědomosti a dovednosti z jiných oblastí informatiky a dalších příbuzných předmětů (např. práce s multimediálním a kancelářským softwarem). Návrh každé počítačové hry je dostatečně přizpůsobený na to, aby jej mohli studenti individuálně a tvořivě rozšířit. Navíc návrh vede ke správnému využití získaných vědomostí.

1.2. Charakteristika předmětu

Předmět je zaměřený na představení inovativního přístupu k výuce programování. Je založený na řešení úloh s využitím myšlenky „zjednodušeného“ OOP. OOP v současnosti dominuje ve vývoji aplikací. Proto je vhodné, aby z této oblasti studenti získali vědomosti a dovednosti. Předmět využívá vývojové prostředí, které nabízí různé formy editace zdrojového kódu (bloková editace pomocí zjednodušené formy a stejně tak reálna editace), což umožňuje vyučovat studenty s různými úrovněmi předchozích znalostí a dovedností. Přehlednost a jednoduchost vývojového prostředí podporuje rychlé a intuitivní pochopení vyučovaných témat. Tím má pozitivní vliv na motivaci studentů.

1.3. Cíl předmětu

Prostřednictví programování interaktivních her v grafickém prostředí student získá vědomosti a dovednosti, které mu umožní:

- identifikovat problém,
- identifikovat vhodné objekty na řešení identifikovaného problému (dekompozice objektů),
- navrhnout třídy objektů, jejich atributy a metody,
- identifikovat a správně využívat vztahy mezi objekty (asociace a dědičnost),
- navrhnout algoritmus na řešení problému a rozdělit ho mezi spolupracující objekty,
- používat prvky zdrojového kódu (větvení, cykly) na implementaci navrženého algoritmu,
- efektivně používat prostředky na ladění zdrojového kódu (angl. debugging),
- vytvořit jednoduchou aplikaci s grafickým rozhraním v prostředí Greenfoot.

1.4. Výsledky vzdělávání

Výsledky vzdělávání dosáhnuté v tomto předmětu je možné shrnout v následujících bodech:

- pochopení základních principů objektově orientovaného programování,
- pochopení základů algoritmizace,
- pochopení syntaxe programovacího jazyka Java,
- schopnost analyzovat program na základě zdrojového kódu,
- schopnost vytvářet vlastní programy s využitím OOP.

1.5. Prostorové a technické požadavky

Počítačová učebna se samostatným pracovním prostorem pro každého studenta a pracovním prostorem pro učitele. Za pracovní prostor se považuje stůl, stolička a osobní počítač (PC), popřípadě notebook. Všechna pracovní místa by měla být připojena k počítačové síti s přístupem na internet (doporučeno).

PC či notebook by měl splňovat tyto minimální požadavky:

- operační systém (alespoň Microsoft Windows 7, Linux (Debian) popř. Mac OS min. 10.10),
- kancelářský softwarový balíček s textovým, tabulkovým a prezentačním editorem (např. Microsoft Office, Libre Office, Open Office, ...),
- Java SE Development Kit (JDK),
- prostředí Greenfoot (alespoň verze 3.8),
- jednoduchý grafický software,
- webový prohlížeč (např. Edge, Google Chrome, Mozilla Firefox, Opera, ...),
- příslušný software pro jiný hardware na PC.

2. Zásady učební osnovy

Navrhovaná učební osnova (angl. syllabus) je navržena tak, aby řešila problémy zjištěné ve výsledcích projektu PR1 a PR2 (pro více informací je třeba projít si kapitolu "Srovnání výsledků s výstupy PR1" ve zprávě PR2). V následující tabulce je uvedený pohled na tvorbu učebních osnov, výsledky vzdělávání, učební materiály a vyučovací aktivity tak, jak byly navržené v PR2. Na základě těchto zjištění poté byly zformulovány zásady učebních osnov.

<table>
<tr><th>Zjištění v rámci PR2</th><th>Zásady učebních osnov formulované v PR3</th></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Na středních školách se měl na začátku výuky algoritmizace představit základní princip OOP. Konkrétně témata pokrývajícími základní koncepty programování. Pokročilejší témata OOP by bylo vhodnější probírat v samostatných kurzech.

Je velmi důležité zajistit a podpořit výměnu informací mezi středoškolskými a univerzitními učiteli za účasti těch zaměstnanců, kteří vytvářejí učební osnovy týkající se programovacích dovedností na všech úrovních vzdělávání.

Z hlediska kurzu, návrhu kurzu, nebo z hlediska učitele by se měly využívat tyto inovativní formy vyučování, resp. výukové metody: smíšené učení (angl. blended learning), učení konáním (angl. learning by doing), řešení problémů, spolupráce na problému, týmová práce, učení založené na problému, aktivní učení a laboratorní učení. Kromě toho by se různé formy inovativních přístupů měly uplatňovat na přednáškách, seminářích a laboratorních cvičeních.

Na motivaci studentů k programování se často používalo programování her a gamifikace. Studentům se líbila možnost být kreativní, resp. soutěžit s ostatními studenty v oblasti vědomostí, pokud byla podpořená vhodným prostředím. Z výsledků přehledu literatury se vybrali čtyři různé typy učení se prostřednictvím her: učení se hraním (angl. learning by playing), učení se tvorbou her, učení se pomocí nástrojů souvisejících s hrami a učení se pomocí gamifikace.</td><td rowspan="2">Učební osnovy musí správně využívat OOP od samého začátku podle přístupu objekty jako první (angl. object first approach). Vhodná úroveň OOP pro střední školy byla identifikována a formulována jako „Light OOP“ (zjednodušené OOP). „Light OOP“ se dá výhodně použít při tvorbě her, protože je jednoduché identifikovat objekty hry stejně tak, jako kompetence a vlastnosti objektů.

Na motivování studentů pomocí her je důležité, probudit v nich zájem o hry. Pro splnění tohoto cíle by se mělo vytvořit několik projektů, ve kterých se vytvářejí hry s různými herními mechanismy.

Navíc několik připravených her umožní:
• vytvořit různé výukové materiály zaměřené na různé podmínky výuky (online/prezenční, intenzivní/celoroční kurz, začátečníci/pokročilí). Toto je klíčová vlastnost pro budoucí výsledky projektu,
• vytvořit jednu hru za přítomnosti vyučujících a další hry jako domácí úlohy (vyučující bude moci zjistit, zda studenti dokážou aplikovat vědomosti v různých souvislostech),
• vytvořit hru podle zadaných pokynů s minimální pomocí vyučujících, aby studenti pochopili důležitost dobře napsaného technického popisu a použili naučené dovednosti na vytvoření hry podle zadání,
• představit nový koncept „Light OOP“ v okamžiku přidávání nových herních mechanismů ve hře. To umožní zastavit vývoj projektu, pokud se vyučující rozhodne pokrýt jen danou podmnožinu „Light OOP“ kvůli specifickým podmínkám.</td></tr>
<tr><td>Při učení a vyučovaní konceptů OOP se ukázalo, že učení se prostřednictvím her má významný vliv na zlepšení dovedností studentů při řešení problémů a též studentům pomáhá jejich zapojení do zábavného prostředí.</td></tr>
<tr><td>Jak zdůrazňují některé výstupy PR2, hlavním cílem by mělo být začlenění učebních a</td><td>Projekty se budou vytvářet na základě principu učení se konáním. Snahou bude minimalizace</td></tr>
</table>

výukových úloh do podnětných a zábavných aktivit, které budou mít pozitivní dopad na vyšší zájem a vyšší míru úspěšného dokončení předmětů týkajících se algoritmizace a programování. Tím by mělo dojít ke zvýšení zájmu středoškoláků o programovaní obecně a v konečném důsledku by to mělo vést lepšímu pochopení konceptů programování a OOP. V takovém případě by studenti nebyli „ztraceni“, při konfrontaci s univerzitními učebními osnovami.	vysvětlování teorie, podpora zkoumání a posílení praktických a tvořivých aspektů učení. Práce ve skupinách povzbuzuje studenty, kteří mohou z počátku mít problémy. Při vývoji nového projektu ve skupině, se mohou použít agilní metody vývoje i vyučování. Před implementací se vyučující může rozhodnout provést analytickou a návrhovou fázi projektu. To umožní využít vědomosti a dovednosti studentů získané v rámci dosavadní výuky při zapojení studentů do tvorby projektu. Studenti mohou být požádáni, aby: • pracovali s informačními zdroji s cílem najít správou hru, • formulovali pravidla hry, respektive požadavky na jejich uplatnění (písemnou či ústní formou), • připravili multimediální obsah (obrázky/zvuky).
Celkovým cílem PR2 bylo najít vhodné inovativní nápady a přístupy k učení a výuce, které by tyto problémy vyřešily. Jak bylo uvedeno v předchozích kapitolách, existuje několik vybraných osvědčených postupů, které by se mohly použít na zlepšení výsledků vzdělávání v době středoškolského studia. Je ale důležité také zmínit, že výsledkem přepracování učebních osnov by mělo být zavedení témat OOP a stanovení cílů při dosahovaní výsledků vzdělávání souvisejících s OOP. Důležitý je také výběr vhodného typu hodnocení: (online) dotazníky jsou jedinou akceptovanou metodou na hodnocení spokojenosti, užitečnosti, zájmu, angažovanosti a zjednodušení konceptů výuky programování a OOP u studentů, které budou definované na středoškolské a nikoliv na univerzitní úrovni.	Se zaměřením na princip učení se konáním jsme se snažili minimalizovat akviziční typy činností a posílit bádavou část. Projekty budou postavené tak, aby umožňovaly jednoduché rozšíření, aby motivovaní studenti byli schopni pracovat na projektech samostatně. Na ověření navrhovaných učebních osnov bude pro každý projekt připraven učební plán. Následně porovnáme analytické výstupy nových projektů využívajících „Light OOP“ s analytickými výstupy učebního plánu vytvořeného pro projekt, který byl vypracovaný v minulosti a je už nasazený v praxi na Slovensku (kromě jiných škol ho na Slovensku využívá i partner projektu Obchodná akadémia Považská Bystrica) a v České republice (využívá ho partner projektu Gymnázium, Pardubice, Dašická 1083). Pozitivní ohlasy na tento ověřený projekt byly zveřejněny v PR2, a proto předpokládáme, že při dodržení stejných osvědčených principů a postupů přeneseme pozitivní přijetí navrhovaného učebního plánu.
Využitím týmové práce v úlohách OOP budou mít studenti možnost podělit se o svoje vědomosti a rozšířit tak implementaci základních konceptů programování na ostatní studenty (peer-to-peer učení).	Při navrhovaní projektů, které zde byly představeny a které jsou založeny na hrách jsme pamatovali na využití různých technik, jako je například EduScrum. Z tohoto důvodu má každý projekt vlastní úložiště ve formě GIT repozitáře a vyučujícím jsou též vysvětleny příslušné náležitosti. I když není nutné využití tohoto
Výsledkem tohoto projektu bude soubor materiálů, které poskytnou vysoce motivovaným	

<table>
<tr>
<td>studentům s dobrými výchozími vědomostmi možnost rozšířit si je prostřednictvím různých aktivit a úloh. Tento typ studentů by mohl výrazně zlepšit výsledky celé vzdělávací skupiny tím, že se podělí o své vědomosti a dovednosti v okamžiku, kdy k tomu dostane příležitost. Další možný přínos tohoto typu studentů pro ně samotné a skupinu je umožnit jim vést týmy.</td>
<td>přístupu (vyučující mohou stále používat tradiční přístup), povede použití agilních technik k výuce:

základů použití verzovacích systémů. Doporučujeme GIT jako nejpoužívanější verzovací systém. Jeho použití usnadňuje

sdílení zdrojových kódů
vývoj nových funkcí pro herní projekty bez dopadu na fázi projektu (např. v podobě samostatné větve), což bude motivovat ambiciózní studenty;

týmové práci - každý student bude zodpovědný za konkrétní aspekt hry, což povede k potřebě efektivní komunikaci mezi členy;
efektivní řízení času (angl. time management) - jednotlivé části projektu budou muset být dodané včas, aby je bylo možné sloučit a pokračovat v další práci, avšak správné používání verzovacího systému a OOP otevírá mnoho možností, jak se vypořádat s časovými problémy, což může vést k pozitivní motivaci studentů v náročných podmínkách.

Zrealizujeme několik školení vyučujících, které budou zahrnovat úvod do verzovacího systému GIT, stejně tak jako využití principů agilního vývoje softwaru ve vyučovaní. Studenti si vytvoří projektové týmy s konkrétními úlohami, na stand-upech si budou vyměňovat nápady, zadávat cíle, realizovat soutěže, vytvářet dokumentaci a další artefakty a prezentovat své řešení.</td>
</tr>
<tr>
<td>Bylo by vhodné podporovat použití vícero různých nástrojů a programovacích jazyků v dřívějších ročnících studia. Koncepty programování by se mohly zjednodušit s využitím vizualizačních nástrojů. Dokonce některé státy už zavedly použití některých nástrojů, jako například Logo nebo Scratch, které jsou zajímavé pro základní školy, ale nikoliv střední školy. Proto by se měly používat pokročilejší nástroje, které jsou určené na podporu OOP. Nástroje Alice a Greenfoot jsou přesně těmi nástroji, které jsou v této oblasti významné.</td>
<td>Používáme prostředí Greenfoot, které využívá programovací jazyk Java. Java je v současnosti velmi populární a v praxi velmi rozšířený programovací jazyk.

Greenfoot navíc obsahuje blokový editor zdrojového kódu pro jazyk Stride. Toto poskytuje možnosti pro vyučující, kteří budou chtít používat prezentované techniky v tomto učebním plánu se studenty mladšího věku.

Greenfoot je velmi vizuální a od začátku umožňuje vytvořit vizualizovaný objekt, který je „živý“ a se kterým možno provádět interakce. Proto je teoretický úvod minimalizovaný a</td>
</tr>
</table>

	studenti mají možnost začít pracovat hned od začátku.
Jak studenti uvedli, při výuce programování je důležité, aby vyučující používali nové a aktuální učební materiály a kreativní vyučovací metody. Stejně tak je potřebná dostupnost a flexibilita učitele při práci se studenty mimo vyučování, aby je motivoval a aby u nich vyvolal větší zájem o předmět.	Na základě prezentovaných principů budou vytvořeny moderní učební osnovy pro vyučující, které budou zahrnovat témata „Light OOP“, které jsou založené na projektové práci a používají koncept object first. Vytvoří se několik projektů založených na hrách, přičemž každý z nich bude dostupný ve formě verzovacího systému - GIT repozitáře. Pro každý projekt bude vytvořený učební projekt, který jej umožní ověřit s už v praxi využívaným a pozitivně přijatým přístupem.

Obsah a rozsah vzdělávacího programu se liší v závislosti od projektu hry, který se v rámci vyučování bude vyvíjet. Podrobnou analýzu najdete v příslušném výukovém designu a v přiložených souborech analýzy.

3. Projekty

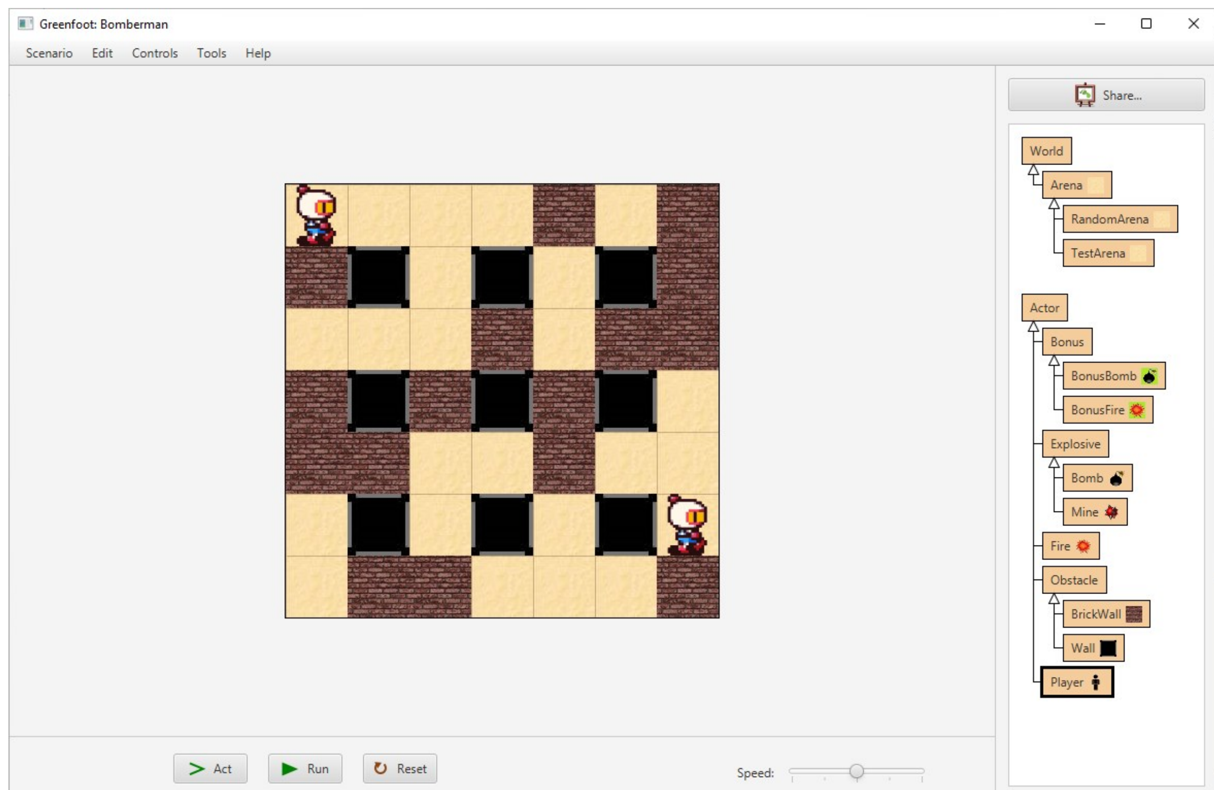
V souvislosti s principy učebních osnov byly vytvořené dva herní projekty, které vhodným způsobem využívají princip object-first a učení se konáním. Navíc jsme zpracovali projekt Bomberman, který byl nosným projektem národního projektu IT Akadémia, který se využívá v praxi na Slovensku. Tento projekt jsme použili na ověření navrhovaných projektů. Organizace všech kapitol projektu je následující.

- **Popis projektu** – základní popis hry se snímkem obrazovky hotové aplikace a souhrnnými pravidly hry. V popisu projektu je uvedené i spojení s tématy „Light OOP".
- **Odkaz na zdrojové kódy** – zdrojové kódy jsou uspořádány formou GIT [1] repozitáře. Použitím korektně zpravovaného repozitáře vyučující přispívá k využití moderního přístupu ve správě zdrojového kódu. Použili jsme verzovací systém GIT jako systém správy verzí, který v posledních letech patří mezi nejpopulárnější [2]. Použiti verzovacího systému GITu navíc umožnuje využití cloudových služeb pro správu repozitářů jako GitHub [3] či GitLab [4], které jsou k dispozici zdarma a poskytují velké množství služeb pro podporu týmové spolupráce. Každý repozitář se řídí následujícími zásadami:
  - **Větev na kapitolu** – úlohy každé kapitoly z učebních osnov jsou vypracované ve specifické větvi. Hlavní větev obsahuje jen inicializační commit (potvrzení) a sloučení větví jednotlivých kapitol.
  - **Commit na úlohu** – každá úloha, která je zaměřená na tvorbu/změnu zdrojového kódu má formu commitu. Popis commitu odpovídá číslu příslušné úlohy.
- **Propojení s návrhem vzdělávání** (angl. learning design) – učební osnovy jsou vytvořené ve formě návrhu vzdělávání. Návrh vzdělávání nám umožňuje definovat výsledky vzdělávání (které pokrývají potřebné kompetence identifikované v PR1 a PR2) a ty pak přiřadit ke kapitolám (všimněte si propojení na větve příslušného GIT repozitáře). Kapitoly jsou uspořádané do jednotek, které se skládají z výukově-vzdělávacích aktivit angl. TLA - Teaching Learning Activities (dále TLA). Všimněte si propojení na commit v příslušném GIT repozitáři. Použití návrhu vzdělávání umožňuje analyzovat rozdělení času, což přímo souvisí s ověřováním deklarovaného principu učení se konáním. Protože návrh vzdělávání byl zpracovaný i pro již zavedený a v pedagogické praxi používaný projekt (Bomberman), umožňuje tento princip identifikovat potenciální problémy v návrhu. Aby bylo možné provést validaci, jsou ostatní projekty uspořádané do kapitol stejným způsobem.
- **Pokrytí témat z „light OOP".**
- **Obsah a rozsah vzdělávacího programu** – přehled zátěže studenta v konkrétním typu učení, jako i přehled přínosu témat k jednotlivým výsledkům vzdělávání.
- **Seznam projektů.** Každý projekt obsahuje:
  - krátký popis,
  - porovnání návrhu vzdělávaní,
  - seznam úloh.

3.1. Bomberman

Bomberman je poměrně známá hra pro více hráčů. Hra se odehrává v aréně, ve které se nacházejí hráči, stejně tak jako některé pevné překážky. Hráč může klást bomby, které po určitém čase vybouchnou. Cílem hry je eliminovat soupeře pomocí bomb. Některé překážky v aréně se dají zničit pomocí bomb. Po zničení překážky se ve hře může objevit náhodný bonus, který například zvyšuje rychlost hráče nebo sílu hráčových bomb. Hra skončí v okamžiku, kdy v ní zůstal jen jediný hráč, který v takovém případě vyhrává. Může se stát, že ve hře nezůstal žádný hráč. V takovém případě hra skončí remízou.

Projekt Bomberman pokrývá většinu témat „light OOP“. Zaměřuje se na hlavní aspekty OOP, které jsou uvedené v níže. Navíc uvádí některá témata přesahující rámec „light OOP“, jako je generování a používání náhodných hodnot s využitím třídy **Random**.



Obrázek 1: Prostředí Greenfoot s konečným stavem projektu Bomberman

Zdrojové kódy jsou k dispozici na:
https://gitlab.kicon.fri.uniza.sk/oop4fun/project-bomberman

Projekt Bomberman je specifický tím, že obsahuje commity, které obsahují práci, která se má vykonat mezi dvěma po sobě následujícími úlohami. Takovéto commity se popisují pomocí zpráv ve tvaru *"Code X.Y"*, kde *X* je číslo kapitoly a *Y* je číslo přechodného úkolu v rámci konkrétní kapitoly. V textu jsou tyto commity označené termínem "přechodový commit".

Návrh vzdělávání je dostupný na:
https://learning-design.eu/en/preview/70bcf65d805b0603f6c1aeab/details

3.1.1. Obsah a rozsah vzdělávacího programu

Celková pracovní zátěž pro studenta je 49h 30min a je rozdělená následujícím způsobem:

49h 30min

Obrázek 2: Pracovní zátěž studenta při řešení projektu Bomberman

Konstruktivní směřování projektu (angl. Constructive alignment) je sumarizované v následující tabulce:

Tabulka 1: Konstruktivní směřování projektu Bomberman

Topic	Assessment		Understanding the basic principles of object-orien... (25)	The ability of creating own programs with the use ... (20)	Understanding the syntax of the Java programming l... (10)	Understanding the basics of algorithmisation (25)	Analysing program execution based on the source co... (20)
	Formative	Summative					
Greenfoot environment	0	0		80%	20%		
Class definition	0	35	60%	20%	20%		
Algorithm	0	20		10%	10%	60%	20%
Branching	0	20		10%	10%	70%	10%
Variables and expressions	0	5		10%	10%	70%	10%
Association	0	10	60%	10%	10%	10%	10%
Inheritance	0	0	50%	30%	10%		10%
Loops	0	40		40%	10%	40%	10%
Lists	0	0		50%	10%	30%	10%
Encapsulation	0	15	50%	30%	10%		10%
Polymorphism	0	15	50%	20%	10%	10%	10%
Random numbers	0	20		30%	10%	50%	10%
Total	**0**	**180**	**270%**	**340%**	**140%**	**340%**	**110%**
	180						

Pro detailnější plán je zapotřebí přejít na kapitolu 5.1.

3.1.2. Rozdělení projektu

Projekt Bomberman je rozdělený do 10 kapitol:

1. Úvod do prostředí Greenfoot 14
2. Algoritmus, ovládací prvky aplikace, vytváření metod 15
3. Větvení a ovládání hráče 16
4. Proměnné, výrazy a pokročilé ovládání hráčů 18
5. Spolupráce objektů a tříd 20
6. Dědičnost a cyklus for 21
7. Seznam a for each cyklus 25
8. Soukromé metody a cyklus while 26
9. Polymorfismus 29

10. Náhodná čísla 31

V projektu jsou aplikování následující témata z „light OOP“:

- třídy, objekty, instance,
- metody, předávání argumentů metod,
- konstruktory,
- atributy,
- zapouzdření,
- dědičnost,
- abstraktní třídy,
- životný cyklus projektu.

1. Úvod do prostředí Greenfoot

Tato kapitola se věnuje základnímu nastavení projektu. Studenti se naučí nastavit rozměry a vzhled prostředí, vytvořit třídu (jako podtřídu třídy **Actor**), vytvořit její instanci, poslat jí zprávu a sledovat její vnitřní stav.

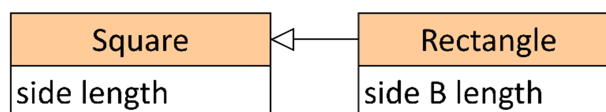
Commit 84de89768134d119dbe94017fe477152c4307b61

1.1. Identifikování objektů

Identifikujte předměty ve svém okolí a uveďte jejich vlastnosti a činnosti, které mohou vykonávat. Dokážete identifikovat objekty, které nemají žádné vlastnosti? Dokážete identifikovat objekty, které nedokážou nic dělat? Dokážete identifikovat nehmotné objekty (takové, kterých se nemůžeme fyzicky dotknout)?

1.2. Ověření hierarchie objektů

Na následujícím obrázku je znázorněná hierarchie tříd **Square** a **Rectangle**. Je to správná hierarchie?



1.3. Vytvoření jednoduché hierarchie

Vytvořte hierarchii tříd pro:

a) dopravní prostředky,
b) zvířata
c) časti počítače.

Uveďte vlastnosti, které definuje každá třída. Které třídy jsou v návrhu nehmotné, tedy se jich nemůžeme dotknout? Takovým třídám budeme v budoucnu říkat abstraktní.

1.4. Vytvoření políčka

V grafickém editoru vytvořte políčko, které bude graficky znázorňovat buňku světa. Vyberte čtvercové zobrazení, ideálně 60x60 bodů. Importujte nebo uložte obrázek do adresáře projektu v adresáři **images** (obrázky). Nastavte obrázek jako obrázek světa. To provedeme pravým kliknutím tlačítka myši na **MyWorld** a volbou „***nastavit obrázek***“. Všimněte si, že třída **MyWorld** dostala v diagramu tříd malou ikonu, která představuje její grafickou podobu.

Commit 5eeecea8bbff323caab71d0a25068d596891d447

1.5. Vytvoření světa a hráče

Upravte konstruktor třídy **MyWorld** tak, aby vytvořil svět s 25x15 buňkami, přičemž každá buňka bude mít velikost 60x60 bodů. Jakým způsobem by bylo potřebné upravit obrázek, aby svět vypadal jako šachovnice (tj. se střídáním různých barevných políček)?

Commit `86da438686edad7900e8e11f595917f49209d346`

Dále vytvořte třídu **Player** (hráč). To se provede pravým kliknutím tlačítkem myši na třídu **Actor** a volbou ***„nová podtřída“***. Podtřídu nazvěte **Player** a vyberte jí obrázek z knihovny. Opět si všimněte malé ikony u třídy **Player** podobně jako u třídy **MyWorld**. Pro vytvoření instance třídy **Player** klikněte pravým tlačítkem myši na třídu **Player**, vyberte ***„new Player“*** a přesuňte ikonu s hráčem na pozadí světa. Instanci umístíte kliknutím na levé tlačítko myši. Vnitřní stav instance zobrazíte tím, že na ni kliknete pravým tlačítkem myši a vyberete *„**prohlížet**“*.

Přechodový commit: `316e48b29455649483c53080c6f5d49d74cf09c3`

1.6. Prozkoumání stavu hráče

Uchopte vytvořenou instanci třídy **Player** myší a přesuňte ji na jiné místo ve světě. Sledujte aktuální náhled jejího vnitřního stavu. Co vidíte? Vytvořte další instanci třídy **Player** a zobrazte také její vnitřní stav. Opět jednu z těchto dvou instancí přesuňte myší. Který vnitřní stav se změnil?

1.7. Interakce s hráči

Zavolejte různé metody poskytované prostředím Greenfoot nad různými instancemi třídy **Player**. To provedete tak, že na instanci kliknete pravým tlačítkem myši vyberete *„**zděděno z Actor**“* a poté např. metodu *„**void move(int)**“* Po výzvě zadáte celé číslo. Sledujte, jak se změní vnitřní stav instance.

2. Algoritmus, ovládací prvky aplikace, vytváření metod

Tato kapitola se zabývá vytvářením veřejných metod, které pohybují s hráčem ve světe. Představuje též nástroje prostředí Greenfoot, které řídí vykonávání scénáře.

2.1. Zápis jednoduchého algoritmu

Napište si postup, jak se připravuje káva, jak se cestuje do školy a jak se vaří oběd.

2.2. Zápis všeobecnějšího algoritmu

Vytvořte všeobecný algoritmus přípravy horkého nápoje. Rozmyslete si, jaké musí být vstupy takového algoritmu, aby byl všeobecný.

2.3. Zkoumání instance třídy

Prozkoumejte metody instance třídy **Player**. To provedete kliknutím na třídu pravým tlačítkem myši a volbou otevřít editor. Co jste zjistili? Analogicky k metodě **act()** přidejte metodu **makeLongStep().**

Přechodový commit: `13b7cca1040b3e1783a8819552d904a05f732817`

2.4. Implementace metody

Do těla metody **makeLongStep()** přidejte takový příkaz, resp. příkazy, aby se instance třídy **Player** posunula o dvě buňky v aktuálním směru. Potom vytvořte více instancí třídy **Player** a na každé z nich zavolejte tuto metodu. Chovají se instance třídy **Player** očekávaným způsobem?

Commit `e678e104ec9b3bac0cc52a11f0a897548cdb5e82`

2.5. Přidání dokumentace

Přidejte dokumentační komentář pro metodu **makeLongStep()**.

Commit `5c04e53c250331bdbf98a01d0c97c722486f8c79`

2.6. *Přidání další dokumentace*

Upravte dokumentační komentář třídy **Player**. Konkrétně přidejte verzi třídy a jejího autora.

Commit 943aedd847dd93796f5a63af824e0556f45b208f

2.7. *Přečteni dokumentace*

Prozkoumejte okno dokumentace.

2.8. *Přidání akce hráče*

Upravte tělo metody **act()** ve třídě **Player** tak, aby se zavolala metoda **makeLongStep()**.

Commit 24fca7f90c940830674d9ee51c28988119a18d83

2.9. *Prozkoumání ovládacích prvků aplikace*

Vyzkoušejte si tlačítka na ovládání aplikace. Vytvořte vícero instancí třídy **Player**. Klikněte na tlačítko ***„Krok"***. Co se stane? Klikněte na tlačítko ***„Spustit"***. Co se stane? Po prvním kliknutí na tlačítko ***„Spustit"*** klikněte na tlačítko ***„Pauza"***, Co se stane? Jaký vliv má posuvník ***„Rychlost"*** na chování metody **act()** po kliknutí na tlačítko ***„Spustit***"? Vyzkoušejte i tlačítko ***„Reset"***.

2.10. *Přidání další akce hráče*

Do třídy **Player** přidejte metodu **walkSquare()**, která bude vykonávat pohyb instancí třídy **Player** do čtverce. Svoji metodu zdokumentujte. Na pohyb a otáčení použijte příslušné metody ze základní třídy **Actor**. Upravte metodu **act()** tak, aby se instance třídy **Actor** při jejím volaní pohybovala do čtverce. Potom ověřte své řešení spuštěním aplikace.

Commit 4721b0e7cceaa883d242d0011485cb9a0ca1d742

3. Větvení a ovládání hráče

Tato kapitola seznamuje s větvením ve formě **if-else** příkazu a **switch** příkazu. Větvení se využívá například při detekci hran světa a kolizí se zdmi, což jsou nové objekty přidané v rámci této kapitoly.

Následující přechodový commit ukazuje úpravu metody **act()** tak, aby se instance třídy **Player** pohybovala o jedno pole a otočila při kontaktu s hranou světa.

Přechodový commit: cd358b49452e8fb3342aba073651fa969a56b275

3.1. *Pohyb hráče*

Upravte kód metody **act()** ve třídě **Player** tak, aby se hráč pohyboval jen po stisku klávesy **M**. Ponechejte kód odpovědný za otočení hráče, když se dostane na okraj světa, ale myslete na jeho umístění. V jaké situaci se může vykonat otočení hráče?

Commit 9d254e0bf2a4c49bd82fd15858d7a3104ab201c7

3.2. *Sledování stavu hráče*

Vytvořte instanci třídy **Player** a umístěte ji do středu hrací plochy. Otevřete okno s vnitřním stavem instance a umístěte ho tak, aby bylo viditelné v průběhu spuštění aplikace. Potom spusťte aplikaci a pozorujte, jak se mění hodnoty atributů **x** a **y** ve třídě **Player**. Jak by se tyto hodnoty měnily při pohybu nahoru, dolů, doleva a doprava? U metody **setRotation()** použijte různé hodnoty (0, 90, 180, 270). Metodu **isAtEdge()** zkuste nahradit metodami **getX()** a **getY()**.

Přechodový commit: f28065cf775055bdd9fc41757af31b818fb76552

3.3. *Přidání detekce okraje světa*

Do těla metody **act()** přidejte kód na správné otočení hráče po dosáhnutí spodního a levého okraje světa.

Commit 1dba08bd12adbf8288087c1957192f8d58745b6e

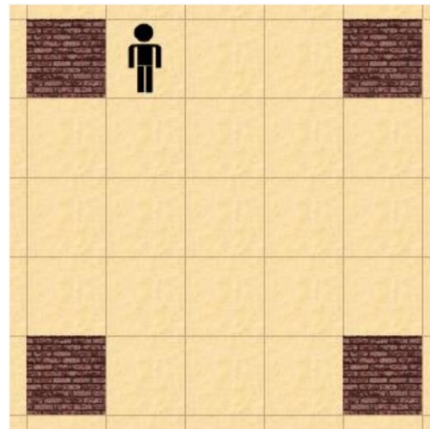
3.4. Přidání překážek

Vytvořte dvě nové třídy jako podtřídy třídy **Actor**. První třídou bude třída **BrickWall** (dále zeď) a druhou třídou bude třída **Wall** (stěna). V grafickém editoru připravte vhodné obrázky s rozměry 60x60 bodů. Potom tyto obrázky přiřaďte k nově vytvořeným třídám.

Commit ce0c50360a562e2a58592ca68b8a2ac83aa98c10

3.5. Sledování pohybu hráče

Vytvořte čtyři instance třídy **BrickWall** a jednu instanci třídy **Player**, jako je znázorněno na obrázku níže. Zkuste odhadnout, jak se bude hráč pohybovat. Spusťte aplikaci. Porovnejte Váš odhad tím, co pozorujete.



3.6. Přidání detekce kolize s překážkou

Do metody **act()** třídy **Player** přidejte kód, aby se zabezpečilo, že se hráč otočí o 90° proti směru hodinových ručiček, v momentě když vstoupí do buňky, která obsahuje instanci třídy **Wall**.

Commit 8dd56c9f4ebb1ece0037eff1a4f8a1efc6425048

V následujícím přechodovém commitu je upraveno chování třídy **Player**. Je využito úplné větvení a přidána další podmínka. Rozmyslete si, jak se zachová instance třídy **Player**, pokud vstoupí do buňky, která obsahuje instanci třídy **BrickWall**.

Přechodový commit: 61415cb24c76a38280e3d82a9a5cf5104d72c5e9

3.7. Předvídání pohybu hráče

Předvídejte, jak se bude pohybovat instance třídy **Player**. Souhlasí výsledek s vaší predikcí?

3.8. Pozorování detekce okrajů

Umístěte jednu instanci třídy **Player** do rohu světa. Předvídejte, jako se bude tato instance pohybovat po spuštění aplikace. Souhlasí výsledek s vaší predikcí?

3.9. Dokončení řešení kolizí se překážkou

Doplňte kaskádu podmínek tak, aby se dotyk s instancí třídy **BrickWall** a třídy **Wall** kontroloval jen pokud se hráč nenachází na okraji světa. Nejdříve implementujte kontrolu s instancí třídy **BrickWall**.

Commit 6f1af5aecda75e6b9a0198b3ee01f22cca4ad182

3.10. Automatický pohyb hráče

Vytvořte metodu na automatický přesun instance třídy **Player**. Veškerý kód z metody **act()** přeneste do nové metody. Identifikátor, resp. název metody zvolte například **moveAutomatically()**.

Commit 52c3706bf61f43e9db82835a5d09274fdff1efc7

3.11. Pohyb hráče pomocí šipek

Vytvořte metodu **moveUsingArrows()** ve třídě **Player**. Naprogramujte tuto metodu tak, aby se hráč pohyboval jen v okamžiku, kdy je stisknutá šipka. Bude se pohybovat v směru stisknuté šipky. Dbejte na to, aby byl kód efektivní. Toto metodu zavolejte v metodě **act()**.

Commit 97e02be7423c9da1ea466547c944b9d58a26fa60

3.12. Připravení obrázků

Připravte pro hráče čtyři obrázky, které se použijí při pohybu hráče nahoru, dolu, vlevo a vpravo. Rozměry obrázků nesmí přesáhnout rozměry buňky, v našem případě 60x60 bodů. Připravené obrázky umístěte do adresáře **images**.

Commit 08df8b3d3224ddead2c54859b3734e5de6455acf

3.13. Použití obrázků

Vytvořte metodu **updateImage()**, která změní obrázek hráče podle jeho aktuální rotace. Do těla metody **act()** přidejte volaní této metody. Následující commit ukazuje možnost využití příkazu **switch** při větvení, jako alternativu **if-else**.

Commit c564ef0bf67120d3b846b0014a5dd1e24c779dec

3.14. Spuštění aplikace

Spusťte aplikaci. Všimněte si, že obrázky se přizpůsobují, ale otáčí se podle toho, jak se otáčí hráč. Vyřešte tento problém.

Commit 81f3be530f0acd454bbe1a0215357628086d547c

3.15. Použití více hráčů

Vytvořte více hráčů a zkuste ovládat je pomocí klávesnice. Co pozorujete?

4. Proměnné, výrazy a pokročilé ovládání hráčů

Tato kapitola se zabývá zavedením proměnných ve formě atributů tříd a parametrů metod. Studenti budou používat atributy na ovládání rychlosti hráče, stejně jako na nastavení konkrétních kláves, kterými se bude ovládat pohyb hráče. To umožní, aby hra měla více hráčů a každý z nich se ovládal jinými klávesami.

4.1. Hledání rozdílu

Jaký je rozdíl mezi následujícími algoritmy?

1.
```
int a;
boolean c;
...
if(a > 0){c = true;}
if(a < 0){c = false;}
```

2.
```
int a;
boolean c;
...
```

```
if(a > 0){c = true;}
else {c = false;}
```

4.2. Zápis jednoduchých výrazů

Napište výraz s využitím logických a relačních operátorů, který vyjádří, že proměnná typu **int** má hodnotu náležící do následujících intervalů: $\langle -10,10)$, $(5,142)$, $\langle -11,3\rangle$, nebo $(1,125\rangle$.

4.3. Vyhodnocování výrazů

Vyberte hodnoty proměnných **x** a **y** a doplňte je do výrazů. Jaké budou hodnoty proměnných **b1** a **b2** v obou příkladech (1 a 2)?

1.
```
int x, y;
...
boolean b1 = x > 0 && y == 1;
boolean b2 = x <= 0 || y <= 0;
```

2.
```
int x, y;
...
boolean b1 = (x > 0) && (y == 1);
boolean b2 = (x <= 0) || (y <= 0);
```

Přechodový commit: 775935e5d09d4db8b5b41ea27d35b0595cfbdb3b

Předchozí přechodový commit upravuje třídu **Player**. Přidává do ní atributy typu **String** (řetězec) **upKey**, **downKey**, **rightKey** a **leftKey**. To budou klávesy, které budou sloužit k ovládání pohybu instance třídy **Player**. Slovo **private** znamená, že atributy budou dostupné jen v rámci třídy **Player**. V rámci této třídy se k atributům bude přistupovat přes slovo **this (**tzn. instance této třídy), tedy **this.upKey**, **this.downKey**, **this.rightKey** a **this.leftKey** Dále je upravena metoda **moveUsingArrows()** tím způsobem, že klávesa **"left"** je nahrazena atributem **this.leftKey**. Podobně jsou nahrazeny ostatní klávesy. Klávesy k ovládání pohybu dané instance je třeba zadat při jejím vytváření. K tomu je třeba vytvořit konstruktor. To je speciální metoda pro vytvoření instance dané třídy, která se provede při jejím vytváření. Konstruktor má opět čtyři parametry - **upKey**, **downKey**, **rightKey** a **leftKey.** Uvnitř konstruktoru je třeba rozlišit mezi **leftKey** a **this.leftKey**. První je parametr konstruktoru a druhý atribut instance dané třídy. Konstruktor není v rámci třídy povinný. Pokud jej neimplementujeme, bude použit výchozí s prázdným kódem.

4.4. Testování konstruktoru

Otestujte konstruktor a upravenou metodu na ovládání pohybu hráče vložením dvou instancí třídy **Player** do světa. Pro každou instanci je nově v dialogovém okně nutné nastavit různé klávesy na ovládání jejího pohybu. Otestujte, zda se vložení hráči dají ovládat nezávisle.

4.5. Přejmenování třídy

Přejmenujte třídu **MyWorld** na **Arena**. Uvědomte si, že název konstruktoru musí být shodný s názvem třídy. Vyzkoušejte si, jak se zachová prostředí Greenfoot, pokud tomu tak nebude.

Commit 7f85bea606e71344e376ab7519be3082fec0ee7c

Doposud bylo nutné vždy manuálně přidávat instance dané třídy, které po kliknutí na tlačítko „***Reset***" zmizely. To je z důvodu, že ve třídě **Arena** nejsou instance vytvořené v konstruktoru. V následujícím přechodovém commitu je vytvořen atribut třídy **Arena player1** typu **Player**. Tím že je **private**, je dostupný jen v rámci třídy **Arena**. Pomocí **new** je pak vytvořen a pomocí metod **addObject** je umístěn na zadanou pozici. Dodejme, že slovo **new** spouští konstruktor dané třídy.

Přechodový commit: d816af6e4f043848d4ef16ac96c80134354d974b

4.6. Přidání dalšího hráče

Přidejte do světa dalšího hráče, například pomocí referenčního atributu **player2**, který bude ovládaný klávesami **w** - nahoru, **s** - dolu, **d** - vpravo a **a** - vlevo. Umístěte hráče na souřadnice **[24,14]**. Po přidání otestujte jeho ovládání.

Commit b69a02f684f702ef963c6adfa08bd8eecc7fd61f

4.7. Referenční atributy

Rozmyslete si, co by se stalo, pokud bychom po vytvoření obou hráčů vykonali přiřazení **this.player1 = this.player2;**? Byl by to problém?

4.8. Rozšíření třídy hráče

Rozšiřte třídu **Player** o další atribut typu **int**, který bude představovat délku kroku hráče. Zde jsou použity dva konstruktory lišící se počtem parametrů. Jedná se o tzv. přetížený konstruktor. Vždy se použije ten, kde je stejný počet parametrů.

Commit 6b4c1680a7ea0b4a690d1671869436ca8e5a27ce

4.9. Integrování délky kroku

Upravte metodu **moveUsingKeys()** tak, aby respektovala nový atribut délky kroku. Vytvořte novou instanci třídy **Player** a otestujte funkčnost programu.

Commit f6bc391828f6b3a08a58d28a9d89b5a7d75eb003

4.10. Umožnění hráčům pohybovat se různou rychlostí

Vašim úkolem je zabezpečit, aby se hráč pohyboval po jednom políčku, ale různou rychlostí. Každý hráč může mít jinou rychlost. Nápovědou pro řešení je možnost naprogramovat různé rychlosti pohybu například tak, že se hráč nebude pohybovat při každém stisknutí klávesy (detekce se vykonává v metodě **act()**, takže když podržíte klávesu, zjistíte jeho stisknutí při každém vykonaní), ale jen při každém **N**-tém stisknutí. Čím větší je **N**, tím menší bude rychlost hráče. Jakým způsobem zadáte **N**? Kde ho uložíte? Jak zjistíte, kolik stisknutí klávesy se už provedlo? (Nápověda: je potřebné počítadlo – atribut **counter** a atribut rychlost - **speed**).

Commit 395bc78a6d243dcd1f471c385e6d010718cc6e44

5. Spolupráce objektů a tříd

Hlavním cílem této kapitoly je spolupráce objektů. V této kapitole studenti do projektu přidají interakci mezi objekty v aréně - například zabezpečí, aby hráč nemohl přecházet přes stěny a zdi. Navíc v této kapitole studenti do projektu přidají objekt bomby - jednu z hlavních částí hry Bomberman.

Přechodový commit: a23230234d7709456beaf7ec1cc3f1c3150fc108

5.1. Přidání kódu pro vertikální pohyb

Analogicky na základě posledního přechodového commitu z poslední kapitoly přidejte kód pro směr nahoru a dolu (změníte hodnotu lokální proměnné **y**).

Commit 369070027d0d2eec7e045181e35a3705a9dda367

Lokální proměnné budou důležité jako parametry metody **canEnter()** připravené v následujícím přechodovém commitu.

Přechodový commit: 3c80533a5a872785cbbd29c3cf1ad4c57f098611

5.2. Ověření možnosti pohybu

Upravte metodu, která zabezpečuje pohyb hráče (**moveUsingArrows**) tak, aby se před změnou polohy hráče ověřila možnost vstupu do cílové buňky.

Commit `4edb51253991482e1ca431fcb158fefa2eac9de2`

V následujícím přechodovém commitu je vytvořena metoda **canEnter()**.

Přechodový commit: `59d511f7170e1c56c97294c65ffbecd32665879d`

5.3. Zohlednění zdí

Upravte metodu **canEnter()** tak, aby hráč reagoval i na zdi - instance třídy **BrickWall** a nemohl přes ně přejít.

Commit `a4757d9ccd7cbc8d5b0241515f268519358e3f61`

5.4. Přidání bomby

Vytvořte novou třídu **Bomb** a navrhněte její atributy, které budou reprezentovat sílu výbuchu. Vytvořte parametrický konstruktor a inicializujte atributy objektu.

Commit: `45fe4733e5ffeb58fee0fe2bc9e1d4af8322b0d4`

Následující přechodový commit přidává do třídy **Player** klávesu pro umístění bomby. Dále přidává atribut pro sílu umístěné bomby (instance třídy **Bomb**) – **bombPower**. Je tedy nutná i úprava konstruktoru.

Přechodový commit: `f014c7b3c13416f0f27ae4cbb4c27b039f23f944`

5.5. Kontrola možnosti umístění bomby

Přidejte do třídy **Player** metodu **canPlantBomb()** s návratovou hodnotou typu **boolean**, která vrátí logickou hodnotu hovořící o tom, či je možné umístit bombu do buňky, ve které hráč právě stojí. Bombu je možné umístit, pokud je stisknutá příslušná klávesa a v buňce se nenachází žádná jiná bomba.

Commit: `ae731b4078f9e4dc19aca31f63badaf068010016`

Následující přechodový commit přidává do třídy **Bomb** atributy časovač (**timer**) a vlastník (**owner**). Časovač reprezentuje dobu, za kterou dojde k výbuchu bomby. Je tedy nutné upravit konstruktor třídy **Bomb**. V metodě **act()** je upraveno chování bomby. Po vypršení časovače bomba vybuchne a zmizí ze světa. Dále je upravena třída **Player**. Je přidán atribut **bombCount**, který reprezentuje počet bomb, které může instance třídy **Player** použít. Při umístění bomby se sníží o hodnotu **1**. při explozi bomby se zvýší o hodnotu **1**. To zařídí metoda **bombExploded().**

Přechodový commit: `fc8486d412a7c621d2a1796979d110b3b9a7167d`

5.6. Přidání zvukových efektů

Rozšiřte hru tak, aby výbuch bomby doprovázel zvukový efekt. Zvuk je možné nahrát nebo stáhnout z internetu. Příkaz na přehrávání zvuku najdete v dokumentaci třídy **Greenfoot**.

Commit: `90da644a58a204eb4de5fb2abab8476c94d73a7c`

6. Dědičnost a cyklus for

Tato kapitola se zabývá základnými principy dědičnosti. Studenti vytvoří předka pro třídy **Wall** a **BrickWall**. Potom vytvoří testovací arénu jako potomka třídy **Arena**. Nakonec se tato část zaměří na cykly s pevným počtem opakovaní.

6.1. *Přidání třídy předka*

Vytvořte třídu **Obstacle** (překážka). Která třída je předkem třídy **Obstacle**? Upravte hlavičky tříd **BrickWall** a **Wall** tak, aby byly potomky třídy **Obstacle**.

Commit: bb77f8893773fec2853c9518fa1025cf2d6d24e2

6.2. *Zjednodušení testu obsazenosti*

Po přidání předka **Obstacle** lze jednodušším způsobem testovat, zda hráč může vstoupit do určité buňky. Ve své metodě **getObjectsAt()** vyžaduje svět jako třetí parametr třídu, kterou má hledat na dané buňce. Protože **BrickWall** i **Wall** jsou **Obstacle**, je možné s nimi zacházet jednotně. Upravte metodu **canEnter()** ve třídě **Player** tak, aby používala jen jeden seznam překážek – instance třídy **Obstackle**.

Commit: c3be615a5a3f5f9992f47068d9c5f6d335e52b93

Následující přechodový commit ukazuje vytvoření třídy **TestArena** jako podtřídy třídy **Arena**.

Přechodový commit: ba2f48e3a8f38503342d26e9dbae6e3a68f1e2ae

6.3. *Úprava třídy Arena*

Upravte třídu **Arena** tak, aby její konstruktor obsahoval dva parametry reprezentující šířku a výšku. Upravte volání konstruktoru předka ve třídě **Arena** tak, aby přebíral tyto parametry. Všimněte si, že **Arena** nemůže být automaticky vytvořená v prostředí **Greenfoot**, protože potřebuje parametry pro konstruktor. Odstraňte z tohoto konstruktoru kód odpovědný za vytváření a umístění hráčů do arény. O to se postarají podtřídy. Z třídy **Arena** můžete také odstranit deklaraci atributů typu **Player**.

Commit: 9477fb9718f0269d4025491d9160ad37da8636b7

6.4. *Oprava třídy TestArena*

Upravte konstruktor třídy **TestArena** tak, aby vytvořil prázdnou arénu s velikostí 7x7 bodů. Na ověření vytvořte instanci třídy **TestArena**. To provedete tak, že z kontextového menu třídy „***TestArena***“ vyberte pravým tlačítkem myši položku „***new TestArena()***“.

Commit: 1a27ff1e964c1a2f5820d14b65bc7850562acd01

6.5. *Přidání informace na rozměr*

Přidejte metodu **showDimensions()** do třídy **TestArena**, která vypíše rozměry arény na obrazovku.

Commit: 49cf3931d68f1d3a18df0a7207a2802b90ef54e5

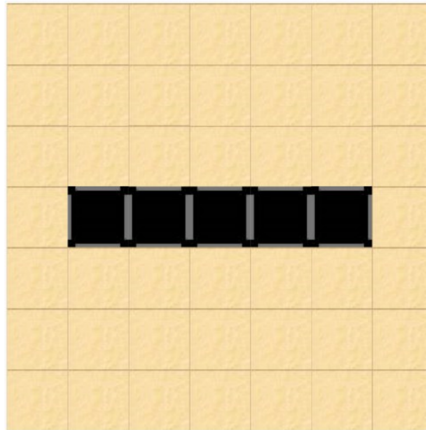
6.6. *Přidání další informace na rozměr*

Do třídy **Player** přidejte metodu **showDimensions()**, která zobrazí rozměry arény, pokud se instance třídy **Player** nachází v testovací aréně – třídě **TestArena**. Použijte metodu **showDimensions()** třídy **TestArena**.

Commit: c5b1e10011483e325271b3a2b101cfbf369f1f44

6.7. *Přidaní stěn do testovací arény*

Upravte konstruktor třídy **TestArena** tak, aby vytvořil arénu s rozměry **7**x**7** bodů, kde stěny jsou rozložené následujícím způsobem:



Připomeňme si, že na vložení instance třídy **Actor** do světa (tj. do potomků třídy **World** a v našem případě třídy **Arena**) můžeme použít metodu **addObject()** třídy **World**, která má tři parametry:

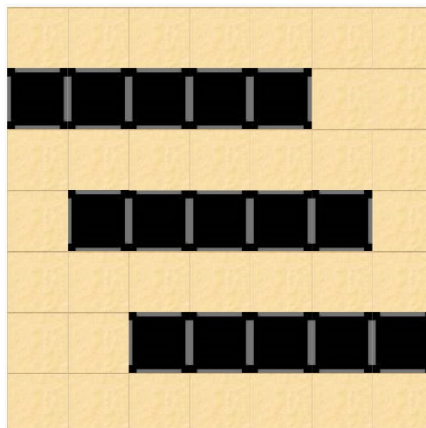
- objekt, který se má vložit a je potomkem třídy **Actor**
- x-ová souřadnice buňky, která se má vložit (tj. index sloupce číslovaný od **0**),
- y-ová souřadnice buňky, která se má vložit (tj. index řádku číslovaný od **0**).

Pozice **[0;0]** ve světe je vlevo nahoře. Jelikož chceme přidat pět instancí třídy **Wall** najednou, použijeme cyklus **for**, který zopakuje v našem případě příkazy uvnitř bloku pro všechna **i=1,2,3,4,5**. Připomeňme, že **super** je volání konstruktoru nadřazené třídy, v našem případě třídy **World**. **Super** musí být v konstruktoru na prvním místě.

Commit: 2edebe2d623173c4a542a80458e98daf9c043173

6.8. Přidání dalších stěn

Upravte konstruktor třídy **TestArena** tak, aby byly buňky rozložené podle následujícího obrázku.



Commit: a9f8c1af0fb0ce2f15ab4267107bdd87f6a76a73

6.9. Zamyšlení se nad tím, jak znázornit více stěn

Zamysleme se nad tím, kolik informací potřebujeme na to, abychom mohli vytvořit libovolný řádek po sobě jdoucích stěn.

6.10. Přidání metody na vytvoření řádku stěn

Vytvořte metodu **createRowOfWalls()** ve třídě **Arena**, která bude mít tři parametry:

- řádek (horní řádek má index **0**), na kterém se mají začít vytvářet steny,
- sloupce (levý sloupec má index **0**), od kterého se mají začít vytvářet steny,

- číslo vyjadřující, kolik po sobě jdoucích stěn se má vytvořit.

Metoda nemá žádnou návratovou hodnotu (v tomto případě používáme klíčové slovo **void**).

Commit: b155b177619e41170ac7759f8d43ebf748a5da6b

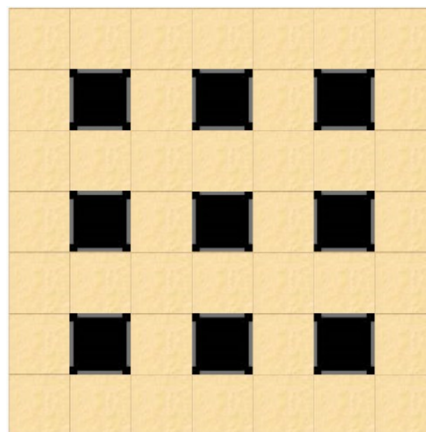
6.11. Použití nové metody

Upravte kód v konstruktoru třídy **TestArena** tak, aby používal metodu **createRowOfWalls()** z předka – třídy **Arena**.

Commit: 91d8888d11fa710c0c2f7eb8a6e1c9da32139a66

6.12. Změna rozmístění stěn

Upravte konstruktor třídy **TestArena** tak, aby vytvořil arénu znázorněnou na obrázku níže. Na tento účel upravte metodu **createRowOfWalls()** tak, aby obsahovala čtvrtý parametr definující rozestupy mezi stěnami.



Commit: 9f6bd3b3187fb8558b68a58ff7b4a3a56f8b1e82

6.13. Zamyšlení se nad tím, jak znázornit obdélník stěn

Zamysleme se nad tím, kolik informací a jaké informace potřebujeme na to, abychom mohli vytvořit stěny v obdélníkovém uspořádání, u kterého bude možné určit počáteční bod a pro které bude možné nastavit vzdálenosti mezi stěnami v řádcích a sloupcích.

6.14. Přidání metody vytvářející obdélník stěn

Ve třídě **Arena** vytvořte metodu **createRectangleOfWalls()**, která bude mít následující parametry:

- řádek (horní řádek má index **0**), od kterého se mají začít vytvářet stěny,
- sloupec (levý sloupec má index **0**), od kterého se mají začít vytvářet stěny,
- počet řádků, které se mají vytvořit,
- počet po sobě jdoucích stěn, které se mají vytvořit v řádku,
- počet prázdných buněk (řádků) mezi řádky,
- počet prázdných buněk mezi stěnami v řádku.

Metoda nemá žádnou návratovou hodnotu.

Commit: 4abb74dc000ce5f91c05b24ee41bace3b0efc395

6.15. Použití nové metody

Upravte konstruktor třídy **TestArena** tak, aby používal metodu **createRectangleOfWalls()** na vytvořeni arény podle úlohy 6.14.

Commit: 69e31efd4dd9c3a4d44925b0a46eac4f693523dd

6.16. Testovaní arény

Vytvořte v aréně několik stěn a přidejte dva hráče. Otestujte funkčnost hry, tj. či hráči nevstoupí do buňky s instancí třídy **BrickWall** nebo **Wall**.

7. Seznam a for each cyklus

Tato kapitola se podrobněji zaměří na seznamy, které se použijí na sledování daných objektů v aréně. Též představí základní metody pro práci se seznamy (vytvoření, přidání prvku, odstranění prvku, přístup k prvku) a též vysvětlí, jakým způsobem používat cyklus **for each** na jednoduchý přístup ke všem prvkům seznamu.

7.1. Ověření ukončení hry

Vytvořte ve třídě **Arena** bezparametrickou metodu **isGameEnded()**, která zjistí, zda hra skončila (zůstal jen jeden nebo žádný hráč), a ve formě návratové hodnoty typu **boolean** oznámí, či se tak stalo. Zatím předpokládáme, že konec hry nikdy nenastane.

Commit: 4c26e33af1a085e947b1271148b012934520ad9d

7.2. Ukončení hry

Přidejte metodu **act()** do třídy **Arena**. V této metodě zkontrolujte, či hra skončila (pomocí metody **isGameEnded()**) a pokud se tomu tak stalo, tak ukončete hru. Pokud chcete zastavit prostředí Greenfoot, použijte příkaz **Greenfoot.stop()**.

Commit: b58443ab6fe8137801a7b2168cdc9935433e15fe

7.3. Přidání seznamu hráčů

Přidejte atribut **listOfPlayers** typu **LinkedList<Player>** do třídy **Arena**. Nezapomeňte, že musíte importovat balík s třídou **LinkedList**. Inicializujte atribut v konstruktoru třídy **Arena**.

Commit: e48d151c44b93fe44a56ad8af3b41d4c21e10432

7.4. Registrace hráčů

Do třídy **Arena** přidejte metodu **registerPlayer()**, která bude mít jeden parametr typu **Player** a vloží ho na konec seznamu **listOfPlayers** pomocí metody **add()**. Upravte potomky třídy **Arena** tak, aby zaregistrovali hráče v předkovi (**Arena**) v okamžiku, když je hráč vložený do světa na správné místo.

Commit: 1166513ff7bfd7d5415715365103821ea0c4691e

7.5. Odregistrování a odstranění hráče

Do třídy **Arena** přidejte funkci **unregisterAndRemovePlayer()**, která bude mít jeden parametr typu **Player**. Metoda odstraní hráče ze seznamu hráčů a potom ho odstraní ze světa.

Commit: f7dadce00ade4d12982a0bce2b68e358f448f01b

7.6. Korektní ukončení hry

Implementujte tělo metody **isGameEnded()** tak, aby metoda vrátila **true**, pokud ve hře zůstane jeden hráč nebo pokud už žádný hráč nezůstane. Použijte vhodné metody seznamu.

Commit: 9ba605f797adec36d7809d6719bc6effa98696c1

7.7. Zjištění hráčů v rozsahu bomby

Upravte kód v metodě **act()** třídy **Bomb** tak, aby před odstraněním bomby ze světa získala seznam všech hráčů, kteří od ní nejsou vzdáleni o víc než daný modifikátor její síly. Metoda

getObjectsInRange() vrací seznam typu **List**. Její první parametr je rozsah – v našem případě modifikátor síly. Její druhý parametr je totožný se třídou, jejíž instance hledá v daném rozsahu.

Commit: c594df2a673abdcadc9ef4161603d54b846e8d62

7.8. Odstranění zasáhnutých hráčů

Pomocí cyklu **for** projděte seznam všech hráčů v seznamu hráčů, které bomba zasáhla. Zrušte registraci těchto hráčů ve třídě **Arena**.

Commit: 2d18cb6f57cf83951cc1a319e5d7f8179508a1a9

Následující přechodový commit ukazuje efektivnější variantu procházení seznamu zasáhnutých hráčů.

Přechodový commit: 850919027622ff95e0985c6eae9409b96337d152

7.9. Úprava třídy Player

Vytvořte metodu **hit()** ve třídě **Player**, která bude vyvolaná bombou hráče po tom, jak ho bomba zasáhne. V této metodě odstraňte hráče ze světa. Upravte kód v metodě **act()** třídy **Bomb** tak, aby odrážel novou funkcionalitu.

Commit: f803a0f8bfddc3ab6f7eef45817d482e11554c63

7.10. Odstranění vlastníka

Vytvořte metodu **removeOwner()** ve třídě **Bomb**, která nastaví jej atribut **owner** na **null**. Instanci objektu si je možno představit jako ukazatele – „šipku" na objekt. Tím, že se nastaví na **null** k objektu neexistuje ukazatel a instance objektu časem zanikne.

Commit: 3a20a51c04cb0ea2b47cf789c2b6503e3ab86eab

V tuto chvíli jsme nastavili atribut **owner** na **null**. Tedy atribut nemusí ukazovat na instanci objetu. Proto je třeba před zavoláním metody **bombExploded()** zjistit, zda existuje její vlastník. Situaci je vyřešena v následujícím přechodovém commitu.

Přechodový commit: a7e97fa06d4c2ab7a39967a80a0410c30bd28707

7.11. Přidání seznamu bomb

Vytvořte atribut **listOfActiveBombs** typu **LinkedList<Bomb>** ve třídě **Player**. Inicializujte ho ve správném konstruktoru. Upravte těla následujících metod podle následujících pravidel:

- v metodě **act**() zaregistrujte nově vytvořenou bombu do seznamu **listOfActiveBombs**;
- v metodě **bombExploded()** odstraňte bombu, která byla zadaná jako parametr (ta, která vybuchla), z atributu **listOfActiveBombs**;
- v metodě **hit()** použijte cyklus **for each** na odstranění vlastníka ze všech bomb v atributu **listOfActiveBombs**.

Commit: 85e552712133e34b0422ebfd4a510147d6d3e027

8. Soukromé metody a cyklus while

Tato kapitola představuje cyklus **while** - cyklus, který se opakuje, dokud je splněna podmínka definovaná na začátku cyklu (tzn. je vyhodnocena jako **true**). Navíc se studenti v této kapitole naučí vytvářet soukromé metody - metody, které může zavolat pouze instance třídy, ve které je metoda definovaná.

8.1. Vytvoření třídy Fire (oheň)

Vytvořte třídu **Fire**. Vyberte vhodné grafické znázorněni. Konstruktor této třídy má jeden parametr, který určuje, jak dlouho bude oheň hořet na místě. Zabezpečte, aby po daném čase oheň ze světa zmizel.

Commit: a68bf80f0af90da351b090217414c63e7fe9492b

8.2. Založení ohně

Upravte existující kód výbuchu bomby tak, aby na místě výbuchu bomby zůstala instance třídy **Fire**. Otestujte svoje řešení.

Commit: ad9cbedf491e77a9d2ea8900bcb3a894b681997c

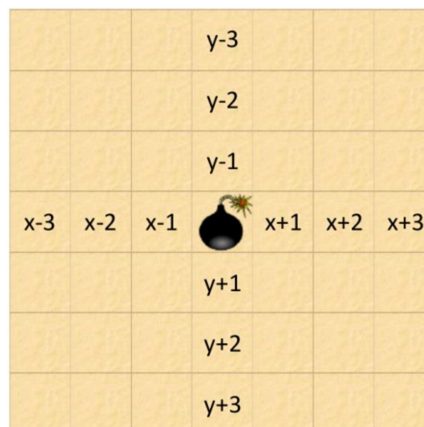
8.3. Rozšíření ohně

Pomocí cyklu **for** rozšiřte výbuch bomby (vytvořte instanci třídy **Fire**) směrem vpravo od bomby. Rozšiřte výbuch na tolik buněk, kolik je uvedeno v atributu síla bomby.

Commit: acf9745908e32a21f43265a4a17662aa06516778

8.4. Rozšířeni ohně do všech směrů

Nastavte výbuch bomby tak, aby vytvořila ohně ve všech směrech (nahoře, dole, vpravo a vlevo). Pomozte si změnou souřadnic, jak je znázorněno na následujícím obrázku.



Commit: 7f1b51fc85583b89e208051958b07d22753977db

8.5. Přepsání cyklů

Přepište všechny cykly **for** pro šíření ohně pomocí cyklu **while**. Druhou část podmínky (zda je možné umístit oheň do další buňky) zatím vynechejte.

Commit: 3e34c8950afe001ceb37e768230a7b91d7ee6cb9

V následujícím přechodovém commitu je vytvořena metoda **spreadFire()** se dvěma parametry. Projděte si metodu a popište, kde založí oheň v závislosti na jejich vstupních parametrech. Metoda je soukromá - **private.** To znamená, že ji lze zavolat jen ze třídy **Bomb**.

Přechodový commit: b11aa272c157dbf3c16f9dd6e8e5fa9ad6542ea1

8.6. Použití soukromé metody

Na rozšíření ohně po výbuchu bomby použijte soukromou metodu **spreadFire()**.

Commit: 9f81a2971a3312ec0883fb2f795d8301b47bb18c

8.7. *Přidání další soukromé metody*

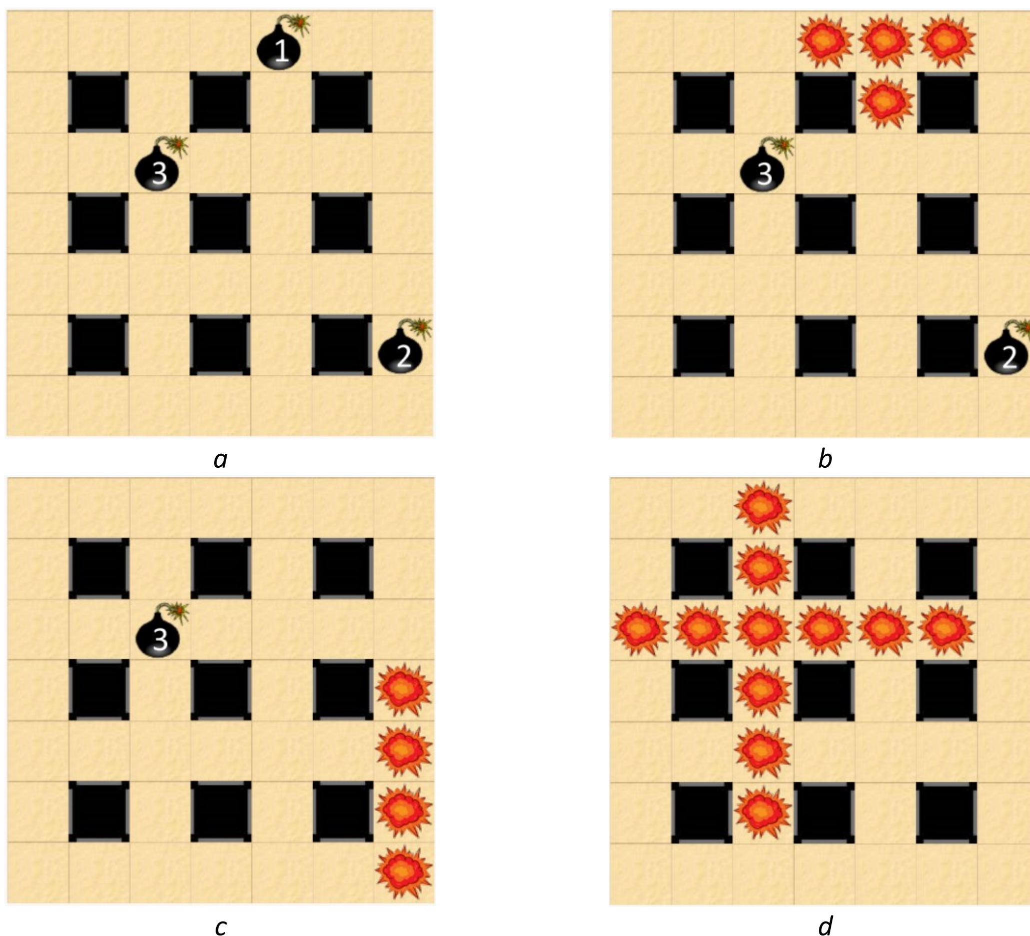
Vytvořte soukromou metodu **canCellExplode()** ve třídě **Bomb**, která jako parametr potřebuje souřadnice řádků a sloupců a vrátí hodnotu **true**, pokud v dané buňce může nastat exploze. Jinak vrátí hodnotu **false**. Šíření ohně nemůže pokračovat pokud:

- dosáhnul okraj světa,
- buňka obsahuje stěnu – instanci třídy **Wall**.

Commit: [53af76996cf6c8e4456627e0cc7bcadd79ab80fa]()

8.8. *Použití soukromých metod*

Pomocí metody **canCellExplode()** nyní můžete upravit podmínku v cyklu **while** v metodě **spreadFire()** ve třídě **Bomb**. Upravte ji v cyklu tak, aby respektovala výsledek kontroly z metody **canBombExplode()**. Otestujte funkčnost řešení pomocí bomby různé sily mezi stěnami – instancemi třídy **Wall**. Testy různých výbuchů jsou znázorněné na následujících obrázcích:



V části ***a*** vidíme původní rozložení, v části ***b*** vybuchla bomba se silou **1**, v části ***c*** vybuchla bomba se silou 2 a v části ***d*** vybuchla bomba se silou **3**.

Commit: [bb7fd73866546b94537195b58b119ab62b31dc9e]()

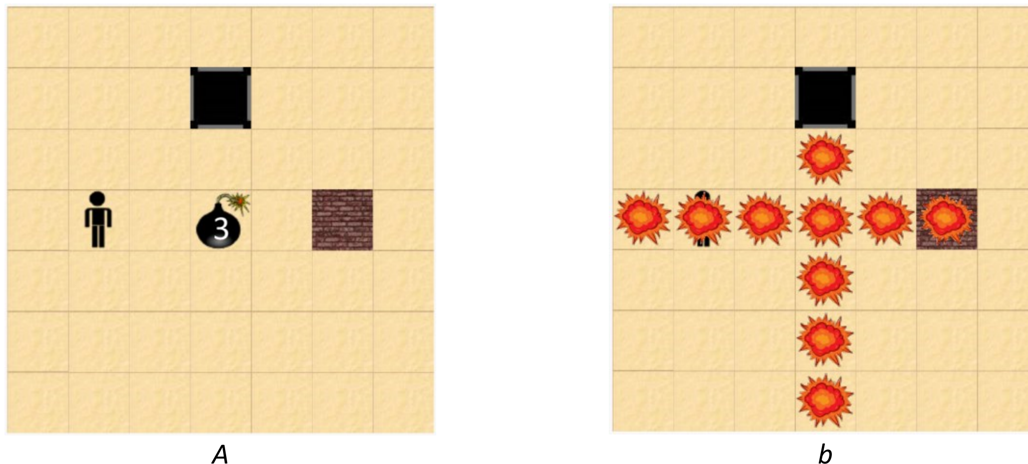
8.9. *Kontrola překážky výbuchu*

Do třídy **Bomb** přidejte soukromou metodu **canExplosionContinue()**, která dostane jako parametry souřadnice řádků a sloupců a vrátí hodnotu **true**, pokud buňka nezastavila výbuch na daných souřadnicích. Pokud buňka zastavila výbuch, metoda vrátí hodnotu **false**. Výbuch nemůže pokračovat, pokud narazil na zeď – instanci třídy **BrickWall**.

Commit: f8d3838b42f0b9ac2984f4275159d5d10a059ebb

8.10. Omezení výbuchu

Pomocí metody **canExplosionContinue()** upravte metodu **spreadFire()** ve třídě **Bomb**. Pokud může výbuch pokračovat z dané buňky, zvyšte hodnotu proměnné **i** o hodnotu **1** a přepočítejte souřadnice nového řádku a sloupce výbuchu. V opačném případě uměle zvyšte hodnotu proměnné **i** na hodnotu větší, než je síla bomby, čímž se cyklus zastaví. Otestujte svoje řešení na situaci z následujícího obrázku:



A *b*

Commit: eeb5e1cb2a887badad6aa8742a0f1fac901ff24a

Jak je vidět na obrázku, zeď exploduje, ale zastaví výbuch. Měla by tedy po výbuchu zmizet. To řeší následující přechodový commit.

Přechodový commit: 91f2be5f9b170744027ce763793d527c4ad9c2c4

8.11. Úprava kontroly přítomnosti ohně

Upravte chování instance třídy **Bomb** tak, aby nevolala metodu **hit()** u hráčů ve svém dosahu při výbuchu. Namísto toho bude hráč sám kontrolovat, či jej střela nepřekrývá. Upravte chování metody **act()** hráče tak, aby se nejdříve zjistilo, či se hráč nepřekrývá s instancí třídy **Fire**. Pokud ano, sám zavolá svoji metodu **hit()**. Tím se zabezpečí, že hráč bude zasáhnutý i ohněm, který hoří po výbuchu bomby.

Commit: 13acf174668443d98eec3de6e68615ee90b4fd9d

8.12. Přidání série výbuchů

Upravte metodu **act()** třídy **Bomb** tak, aby bomba vybuchla i v situaci, kdy se bomba nachází v té samé buňce jako oheň. Řešení ověřte vykonáním řetězové reakce několika bomb.

Commit: 137f5de9a734561a11ea0afc6cfd5dde5e1ee1bd

9. Polymorfismus

Cílem této kapitoly je naučit studenty vytvářet virtuální metody a podle potřeby je překrývat v potomcích. Studenti se také seznámí s novou viditelností atributů a metod – modifikátorem přístupu **protected**. Studenti využijí polymorfismus na zjednodušení existujícího kódu.

9.1. Přidání min

Začněte přidáním miny - třídy **Mine**. Mina vybuchne, pokud na ni vstoupí hráč. Mina také vždy vybuchne, pokus ji zasáhne oheň (např. z bomby, která vybuchla v blízkosti). Na svém místě zanechává oheň (jen tam, oheň se nešíří). Přidejte možnost, aby hráč mohl položit miny (podobně jako bomby) po stisknutí klávesy (např. **control** nebo **shift**). Podobně jako při bombách, má i zde hráč omezený počet min (tj. pokud položí všechny miny, další minu může položit jen pokud jedna z předtím

položených min vybuchla). Počáteční počet min je nastavený parametrem v konstruktoru třídy **Player**. Pokud chcete ve třídě **Player** zaregistrovat miny a reagovat na jejich výbuch, postupujte podobným způsobem jako při bombách (vytvořte seznam min, přidejte metody **mineExploded()**, **canPlantMine()** atd.)

Commit: 5627d97dff9b5d5c1533d9cfb1a3f4a7c8bf631d

9.2. Přidání předka

Vytvořte společného předka pro třídy **Bomb** a **Mine** – třídu **Explosive**. Které atributy a metody by se měly přesunout do předka a které by měly zůstat v potomcích? Upravte existující třídy podle svého návrhu.

Commit: 48eb9fb40e332e027c2e2b168a1d6d3149d7fd3d

9.3. Úprava viditelnosti atributu

Upravte viditelnost atributu **owner** ve třídě **Explosive** na **protected**. Viditelnost atributu **private** by totiž umožnila použít jej pouze uvnitř třídy **Explosive** a nikoliv je jejích potomcích. Parametr **protected** znamená viditelnost v rámci balíčku, což je v našem případě v rámci projektu hry Bomberman.

Commit: 5a0b733d1df877777bad7688631f1500cd8c7041

9.4. Přidání textového výstupu

Vytvořte metodu **printWhoYouAre()** bez parametrů ve třídě **Explosive**, která nemá návratovou hodnotu. Metoda vypíše text *"EXPLOSIVE"* na obrazovku, na místo, kde výbušnina umístěna. Vytvořte instanci třídy **Mine** a zavolejte metodu **printWhoYouAre()**. Co se stane? Vytvořte instanci třídy **Bomb** a zavolejte metodu **printWhoYouAre()**. Co se stane v tomto případě?

Commit: 0528c0a1ec75f08b2f2d088314432c5169448169

9.5. Zlepšení textového výstupu

Vytvořte metodu **printWhoYouAre()** ve třídě **Mine** se stejnou hlavičkou jako ve třídě **Explosive** (tj. metoda bude mít stejný název, stejné parametry a stejný typ návratové hodnoty). Metoda vypíše na obrazovku text *"MINE"*. Opět vytvořte instanci třídy **Mine** a třídy **Bomb**. Pokuste se uhodnout, co se stane, když zavoláte testovací metodu v instanci třídy **Mine** a v instanci třídy **Bomb**. Potom zavolejte metody. Odpovídá váš odhad výsledku?

Commit: 2d3e1fe13a8d627730bf01a729ae7c1619eaf64b

9.6. Dokončení textového výstupu

Překryjte metodu **printWhoYouAre()** ve třídě **Bomb** tak, aby na obrazovku vypisovala text *"BOMB"*. Ověřte správnost svého řešení.

Commit: 09e622e84efe580fed9c0b67f2bdd2a48cd06b1f

9.7. Přidáni zpracování výbuchu

Vytvořte metody **shouldExplode()** a **explosion()** ve třídě **Explosive** a metodu **explosiveExploded()** ve třídě **Player** tak, jak je popsané výše. Těla metod zatím neimplementujte. Pokud bude zapotřebí návratová hodnota, použijte **false**.

Commit: 749e2e91092f8929877af7d8aeb068f010b110ed

9.8. Umožnění exploze výbušniny

Napište tělo metody **act()** ve třídě **Explosive**.

Commit: 996cfa17d46269663becddd1e1fbe2fb82280370

9.9. Umožnění výbuchu bomby

Překryjte metody **shouldExplode()** a **explosion()** ve třídě **Bomb**. Použijte příslušný kód z metody **act()** ve třídě **Bomb**. Všimněte si, že je možné jednoduše napsat těla metod, protože není zapotřebí uvažovat nad podmínkami (to je provedeno v nadřízené třídě).

Commit: c1eed984efd956ff4ddd914288201773afd92d44

Podobně jako se u konstruktoru použilo **super** pro volání nadřazeného konstruktoru, tak je toto možné i u ostatních metod. Následující přechodový commit ukazuje zavolání metody **act()** ve třídě **Bomb** ze třídy **Explosive** s využitím **super**.

Přechodový commit: 5a6d0ceaa51024854ac579f1c3a888309914e22a

9.10. Umožnění výbuchu miny

Překryjte metody **shouldExplode()** a **explosion()** ve třídě **Mine**. Použijte příslušný kód z metody **act()** ve třídě **Mine**. Proč je zapotřebí na konci odstranit metodu **act()**?

Commit: f3d63427a9a975f5205e483ea535c4440582de28

9.11. Přidání interakce s ohněm

Upravte tělo metody **shouldExplode()** ve třídě **Explosive** tak, aby metoda vrátila **true**, pokud se instance dotkne instance třídy **Fire**. Upravte nadřazené metody **shouldExplode()** ve třídách **Bomb** a **Mine** tak, aby používali funkčnost metody předka.

Commit: 82767b061d4e2e48810f57133a090c2b8d98017d

9.12. Zjednodušení atributů hráčů

Odstraňte atributy **listOfActiveBombs** a **listOfActiveMines** ze třídy **Player**. Přidejte do třídy **Player** jeden atribut typu **listOfActiveExplosives** typu **LinkedList<Explosive>**. Inicializujte ho v konstruktoru a odstraňte inicializaci původních atributů z konstruktoru.

Commit: c174bfb8c5512cb2e9e1fb45ecf36c83d84b5af7

V souvislosti s předchozí změnou je ve třídě Player zapotřebí provést úpravy všude tak, kde se používaly seznamy **listOfActiveBombs** a **listOfActiveMines.** Je nutné je nahradit seznamy **listOfActiveExplosives.** Vše je vyřešeno v následujícím přechodovém commitu.

Přechodový commit: ce7945c389bfd55b9eabf76c4599bc83f273e07e

9.13. Zjednodušení zacházení s výbuchem

Implementujte tělo metody **explosiveExploded()**. Použijte operátora **instanceof** na určení, zda je výbušnina **Bomb** nebo zda je výbušnina **Mine**. Na základě jejího skutečného typu zvyšte počítadlo dostupných bomb nebo počítadlo dostupných min. Nezapomeňte odstranit výbušninu ze seznamu aktívních výbušnin. Nakonec odstraňte nepotřebné metody **bombExploded()** a **mineExploded()**.

Commit: 7a50f76f5de55cad056d372563214f0cbf581c97

10. Náhodná čísla

Tato kapitola se věnuje náhodnosti. Studenti se seznámí se třídou **Random**. S využitím jejich instancí budou generovat náhodná čísla. Kapitola také ukazuje způsob generování náhodných čísel bez použití třídy **Random**, a to přímo s využitím prostředí **Greenfoot**. Studenti použijí náhodná čísla na náhodné uspořádání arény a na přidávání bonusů do světa – speciálních prvků, které se vytvoří po výbuchu zdí a které zlepšují vybrané vlastnosti hráčů.

10.1. *Zamyšlení se nad náhodností*

Přemýšlejte o tom, co je to náhodnost, jak můžeme získat nějaký náhodný výsledek z experimentu a jaké náhodné jevy pozorujeme ve světe okolo nás.

10.2. *Zamyšlení se nad generováním náhodných hodnot*

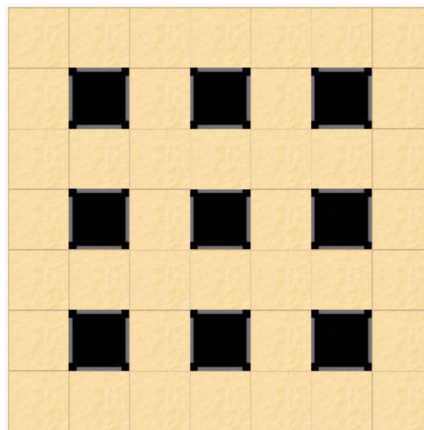
Uvažujme o klasické kostce se šesti stranami. Mohli bychom ji použít na generování náhodné pozice na šachovnici o velikosti 6x6 políček? A co na šachovnici s 3x3 políčky? Jak by se změnil způsob generování, kdybychom použili minci? Navrhněte algoritmy na generování pozicí.

10.3. *Pozorování náhodnosti*

Pomocí algoritmu z předchozí úlohy a pomocí kostky vytvořte náhodné pozice na šachovnici. Výsledky zaznamenejte na šachovnici. Dá se ve výsledcích pozorovat nějaká pravidelnost?

10.4. *Připravení náhodné arény*

Připravte arénu. Vytvořte potomka třídy **Arena**, který pojmenujete např. **RandomArena**. V konstruktoru nastavte příslušnou velikost světa. Doporučeno je pravidelné rozložení stěn s jedním prázdným políčkem mezi stěnami, jak je znázorněné na následujícím obrázku:



Commit: f937ffbd36f64b0df7c9d904e974a2437d9c8959

V přechodovém commitu je připravena zatím prázdná metoda **createRandomWall()**.

Přechodový commit: f692394053ce0a3f76c6ec294c977ef3d7dba91c

10.5. *Přidání generátoru náhodných čísel*

Přidejte referenčního atributu typu **Random** do třídy **RandomArena**. Nezapomeňte, že třída **Random** je definovaná v balíku **java.util.Random**. Atribut inicializujte v konstruktoru.

Commit: 69655b53dac3fe0565d88e124cab63a03ee5b533

V následujícím přechodovém commitu jsou vygenerována náhodná čísla a uložena do proměnných **randomColumn** a **randomRow**.

Přechodový commit: 1ba83506b6b2ea5b81ba79c67c506a81d11e8dc5

10.6. *Kontrola obsazenosti buněk*

Do třídy **RandomArena** přidejte soukromou metodu **isCellFree()**, která bude požadovat dva parametry – sloupec a řádek. Metoda vrátí **true**, pokud je buňka ve světě volná (neobsahuje žádnou instanci třídy **Actor**). V opačném případě vrátí **false**.

Commit: 05dc486ffc38a6dd69e510072ce15d0e9ad4698a

Nyní již lze vygenerovat souřadnice buňky, kde není žádný objekt třídy **Actor** a vložit do ní instanci třídy **BrickWall**. Vše je vyřešené cyklem **while** v následujícím přechodovém commitu.

Přechodový commit: 6507a2386b739c1894f96cba24d49661d0195571

10.7. Generování náhodných stěn

Upravte konstruktor třídy **RandomArena** tak, aby náhodně generoval stěny do třetiny počtu všech buněk arény.

Commit: e92647d730fb61f573a144c06c690d9366889869

10.8. Přesun náhodnosti do předka

Metody **createRandomWall()**, **isCellFree()** a atribut generátoru (včetně jeho inicializace v konstruktoru) přesuňte do předka **Arena**. Nezapomeňte přesunout i řádky importu.

Commit: bd5f0d27c3437995362161e948dbf2af381e8715

10.9. Zevšeobecnění náhodného generování

Do metody **createRandomWall()** přidejte parametr typu **Actor** – bude to objekt, který vložíme na náhodné souřadnice. Změňte název metody (např. **insertActorRandomly()**) a aktualizujte volání metody ze třídy **RandomArena**.

Commit: 55cf63e631abcc7924e83fa38d558e3bd8e1f8aa

10.10. Přidání bonusů

Vytvořte třídu **Bonus** jako potomka třídy **Actor**. Vytvořte dvě podtřídy třídy **Bonus** – třídu **BonusFire** a třídu **BonusBomb**. Nastavte pro tyto třídy vhodné obrázky.

Commit: df2b5f48f12f9cdc1de0528422dbda07e5051d59

10.11. Náhodné vytváření bonusů

Upravte kód v metodě **act()** třídy **BrickWall**. Po zničení instance třídy **BrickWall** na jejím místě vygenerujte s pravděpodobností 10 % bonusový oheň a s pravděpodobností 10 % bonusovou bombu. V 80 % případů nic nevygeneruje.

Commit: 7ec336a713814cdae7fe1be841c5e4feacd75e74

10.12. Uplatnění bonusu

Připravte třídu **Bonus**. Vytvořte metodu **protected applyYourself()** bez návratové hodnoty, která má jeden parametr typu **Player**. Tuto metodu nechte ve třídě **Bonus** prázdnou. Potom v metodě **act()** definujte akci, která nejdříve zjistí, zda hráč vstoupil na bonus (metoda **(Player)this.getOneIntersectingObject(Player.class)**), a pokud ano, aplikuje ho (voláním metody **applyYourself()**). Nakonec odstraňte bonus ze světa.

Commit: 684b7d72b6d5c22a355ae7ca76e53f8cc540908d

10.13. Zvýšení počtu bomb

Přidejte do třídy **Player** metodu **public increaseBombCount()** bez parametru, která nemá návratovou hodnotu a která zvýší počet bomb, které muže hráč položit, o jednu. Potom přepište metodu **applyYourself()** ve třídě **BonusBomb**. Uplatnění bonusu znamená zvýšení počtu bomb, které má hráč k dispozici, o jednu (volaní metody **increaseBombCount()**).

Commit: cb43a15261ded0f12df3b92cfe5d50efe24e9e95

10.14. Zvýšení síly bomby

Přidejte do třídy **Player** metodu **public increaseBombPower()** bez parametru, která nemá návratovou hodnotu a která zvyšuje hodnotu atributu **bombPower** o hodnotu jedna. Potom přepište metodu **applyYourself()** ve třídě **BonusFire** a zvyšte sílu bomby hráče o hodnotu jedna.

Commit: `4abf63b01f1eee1c870821a90cb0e920594443af`

3.2. Tower defense

Ve hrách typu „*Tower defense*“ (Obrana věže) hráč používá věže, které střílejí určitý druh střel, aby zabránily nepřátelům dosáhnout a zničit cílové místo (označované angl. *orb*). Nepřátelé vždy používají stejnou cestu, avšak jak hra pokračuje, nepřátelé jsou silnější a objevují se ve větších skupinách. Hráč musí umístit věže na strategické místa, aby je zastavil v následujících vlnách. Existuje mnoho verzí hry, které se odehrávají v různých světech s použitím různých entit (od balónů přes skřety, obranu pomocí věží podobných zvířatům až po magické bazény).

Tento projekt představí jeden typ věže, kterou hráč může nebo nemůže ovládat ručně. Nepřátelé budou různého typu s rozdíly v jejich indexu zdraví (angl. *health points*, zkráceně HP) a rychlosti. Představený herní design je lehce rozšířitelný, ponechává dostatek prostoru jak pro kreativitu studentů, tak pro zadání vyučujícího. Z tohoto důvodu jsme se v prvních kapitolách snažili minimalizovat aktivity hodnotícího typu. Navíc návrh ponechává dostatek prostoru na přirozené a jednoduché zavedení témat i mimo rozsah „light OOP“ (jako je polymorfismus). Projekt v jeho konečném stavu po dobu hry je znázorněn na obrázku 3.

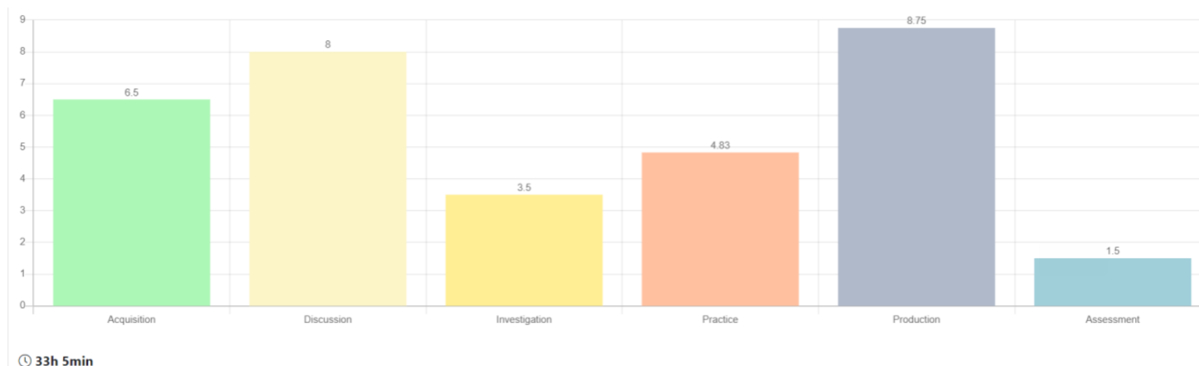
Obrázek 3: Prostředí Greenfoot s konečným stavem projektu Tower defense

Zdrojové kódy jsou k dispozici na:
https://gitlab.kicon.fri.uniza.sk/oop4fun/project-tower-defense

Návrh vzdělávaní je dostupný na:
http://learning-design.eu/en/preview/452257b563cbf14b6f06acfd/details

3.2.1. Obsah a rozsah vzdělávacího programu

Celkové pracovní zatížení studenta je 33 h 5 min a je rozdělené následujícím způsobem:



Obrázek 4: Pracovní zátěž studenta při řešení projektu Tower defense

Konstruktivní směřování projektu je sumarizované v následující tabulce:

Tabulka 2: Konstruktivní směrováni projektu Tower defense

Topic	Assessment		Understanding the basic principles of object-orien... (25)	Understanding the basics of algorithmisation (25)	Understanding the syntax of the Java programming l... (10)	Analysing program execution based on the source co... (20)	The ability of creating own programs with the use ... (20)
	Formative	Summative					
Greenfoot environment	0	0					100%
Class definition	0	0	60%		20%		20%
Algorithm	0	0		60%	10%	20%	10%
Branching	0	0	10%	60%	10%	10%	10%
Variables and expressions	0	0	40%	30%	20%		10%
Association	0	60	30%	30%	10%		30%
Inheritance	0	30	40%	20%	10%		30%
Encapsulation	0	0	50%	10%	20%		20%
Total	**0**	**90**	**230%**	**210%**	**100%**	**30%**	**230%**
	90						

Pro detailnější plán je zapotřebí přejít na kapitolu 5.2.

3.2.2. Rozdělení projektu

Projekt Tower defense je rozdělený do sedmi kapitol:

1. Úvod do prostředí Greenfoot 37
2. Algoritmus, ovládací prvky aplikace, tvorba metod 39
3. Větvení a ovládání nepřítele 40
4. Proměnné a výrazy 42
5. Asociace 44
6. Dědičnost 48
7. Zapouzdření, statické metody a atributy 51

Aplikovaná témata z light OOP:

- třídy, objekty, instance,
- metody, předávání argumentů metod,
- konstruktory,

- atributy,
- statické proměnné a metody
- zapouzdření,
- dědičnost,
- abstraktní třídy,
- životný cyklus projektu.

1. Úvod do prostředí Greenfoot

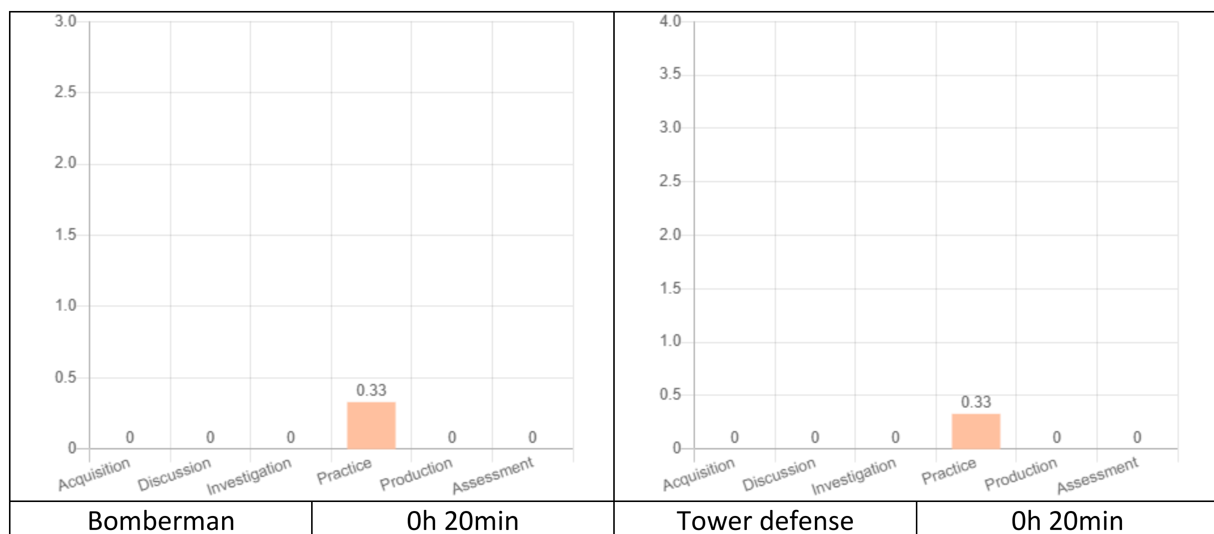
Tato kapitola je věnována vytvoření projektu. Studenti budou schopni vytvořit nový projekt v prostředí Greenfoot, vytvořit třídu (jako potomka třídy **Actor**), vybrat obrázek pro nově vytvořenou třídu, vytvořit její instanci a poslat jí zprávu.

Vytvořte nový projekt. Dejte mu správný název (např. **TowerDefense**) a uložte ho na správné místo.

Commit: `9046f5353d857dcc112abd92d7b7170abcc64a80`

V tabulce 3 je shrnuté porovnání pracovní zátěže kapitoly úvod do prostředí Greenfoot mezi projekty Bomberman a Tower defense. V návrhu není žádný rozdíl.

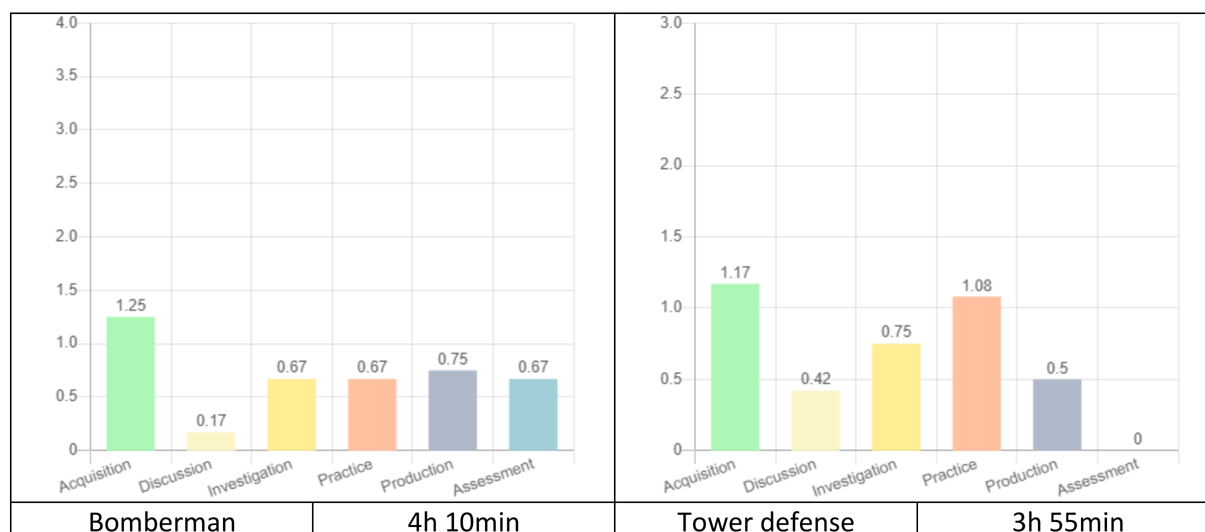
Tabulka 3: Porovnaní pracovní zátěže kapitoly úvod do prostředí Greenfoot mezi projekty Bomberman a Tower defense



Bomberman	0h 20min	Tower defense	0h 20min

V tabulce 4 je shrnuté porovnaní časové zátěže kapitoly definice tříd mezi projekty Bomberman a Tower defense. Pracovní zátěž projektu Tower defense je nižší, více prakticky orientovaná s důrazem na zkoumání a procvičování.

Tabulka 4: Porovnaní pracovní zátěže kapitoly definice tříd mezi projekty Bomberman a Tower defense



Bomberman	4h 10min	Tower defense	3h 55min

1.1. *Úloha 1.1 Identifikovaní objektů z projektu Bomberman.*

1.2. *Příprava světa*

Upravte zdrojový kód třídy **MyWorld** (dvakrát na ni klikněte) a vytvořte svět s velikostí 24x12 buněk. Každá buňka by měla mít velikost 50 bodů.

Commit: a593cd4a92d0fa0db78275614c3e41a2e96b4e57

1.3. *Příprava obrázku pro svět*

Najděte nebo vytvořte vhodný obrázek pro pozadí světa. Můžete použít buď připravené obrázky (vyberte položku ***„Set image“*** z kontextového menu třídy ***„MyWorld“*)**, nebo vlastní obrázek (v tomto případě zkopírujte obrázek do podadresáře **images** v adresáři projektu a vyberete ho stejným způsobem, jak bylo popsáno předtím).

Jako pozadí můžete použít jediný obrázek, který pokryje celou plochu světa (vypočítejte potřebnou velikost obrázku vzhledem k velikosti světa), nebo menší obrázek, který se bude opakovaně kopírovat (použijte čtvercový obrázek s velikostí jedné buňky).

Commit: 1184980643db082cfdd6bde9984bceaddf010d49

1.4. *Vytvoření třídy Enemy*

Vytvořte nepřítele. Nepřítel bude postupovat směrem k cílovému místu, aby ho poškodil a nakonec zničil. Vytvořte nového potomka třídy **Actor** (z kontextového menu třídy „***Actor***“ vyberte položku „***New subclass***“). Dejte mu správné jméno - **Enemy** a obrázek.

Commit: 4981400623729c3d112b54454b6e6151e18426bf

1.5. *Vytvoření instance třídy Enemy*

Vytvořte instanci třídy **Enemy** (z kontextového menu třídy „***Enemy***“ vyberte položku „***new Enemy()“***, umístěte instanci do světa kliknutím levým tlačítkem myši na požadovanou pozici). Prozkoumejte její vnitřní stav (z kontextového menu vytvořené instance vyberte položku „***Inspect***“).

Vytvořte další instanci třídy **Enemy** a umístěte ji na jinou pozici. Porovnejte vnitřní stavy obou vytvořených instancí.

1.6. Odesílání zpráv instancí

Pošlete zprávu instanci třídy **Enemy** (z kontextového menu vybrané instance vyberte položku „***zděděnou z třídy Actor***“ a potom vyberte požadovanou položku). Co se stalo? Jak byl ovlivněný vnitřní stav příslušné instance?

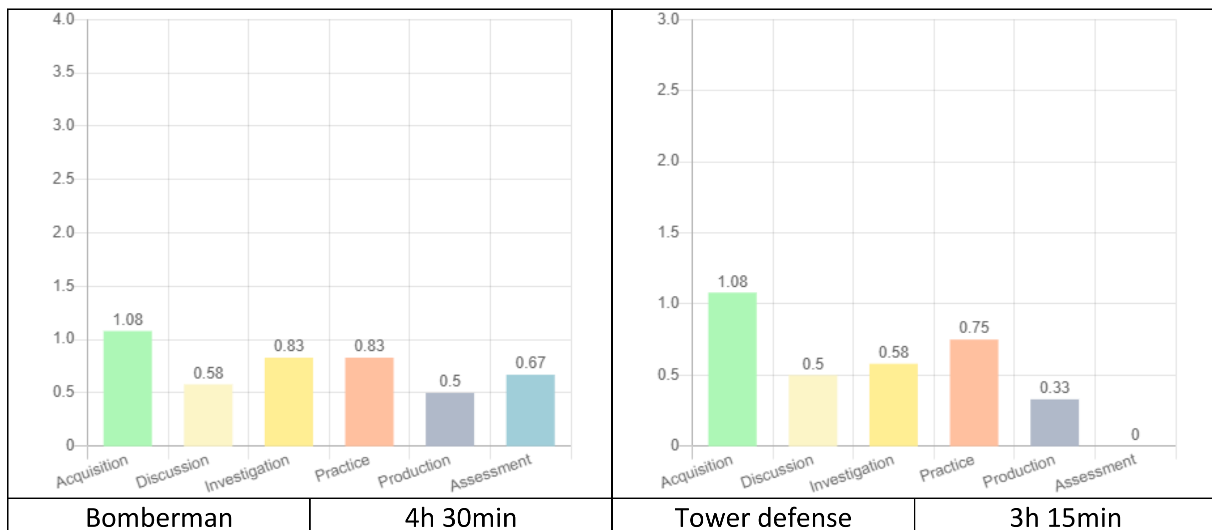
Pošlete zprávu instanci třídy **Enemy**, aby se přesunula na pozici **[12, 6]** a aby byla otočená směrem dolů. Napište posloupnost odeslaných zpráv na papír.

2. Algoritmus, ovládací prvky aplikace, tvorba metod

Tato kapitola se zabývá základy algoritmizace a představuje základy práce s dokumentací. Studenti budou umět zavolat metodu ve zdrojovém kódu, napsat a zobrazit dokumentaci.

V tabulce 5 je shrnuté porovnaní pracovní zátěže kapitoly algoritmizace mezi projekty Bomberman a Tower defense. Projekt Tower defense je podobný projektu Bomberman, avšak s výrazně nižším počtem TLA typu zkoumání. Je vidět, že mnohé TLA jsou stejné jako v projektu Bomberman. Je to proto, že v tomto stavu projektu se ve velké míře překrývá to, co se s nimi dá dělat. Je možné najít inspiraci v úlohách projektu Bomberman, pokud bude potřebné posílit část zkoumání v učebních osnovách. Pro potřeby výuky témat „light OOP“ s využitím projektu Tower defense však považujeme navrhnutý počet TLA úloh typu zkoumání za dostatečný.

Tabulka 5: Porovnaní pracovní zátěže kapitoly algoritmizace mezi projekty Bomberman a Tower defense



Bomberman	4h 30min	Tower defense	3h 15min

2.1. *Úloha 2.1 Napsaní jednoduchého algoritmu z projektu Bomberman.*

2.2. *Úloha 2.2 Napsaní všeobecnějšího algoritmu z projektu Bomberman.*

2.3. *Zavolaní metody*

Do těla metody **act()** přidejte příkaz, aby se instance třídy **Enemy** posunula o dvě buňky v aktuálním směru. Potom vytvořte další instance třídy **Enemy** a zavolejte metodu pro každou instanci. Chová se instance třídy očekávaným způsobem?

Commit: 7ba327ebeba6a13be68d9d21cc7e74b0da376132

2.4. *Přidání dokumentace*

Přidejte dokumentační komentář k metodě **act()**.

Commit: 68b1c82c7df2c7826f2d3f78373498569adab7e9

2.5. *Přidáni další dokumentace*

Upravte dokumentační komentář ke třídě **Enemy**. Přidejte verzi třídy a jejího autora.

Commit: [1a7a9f83c5271a7c0dfa46ce3b1ee65682b0c5e5]

2.6. *Úloha 2.7 Pročtení dokumentace z projektu Bomberman*

2.7. *Úloha 2.9 Prozkoumání ovládacích prvků aplikace z projektu Bomberman*

3. Větvení a ovládání nepřítele

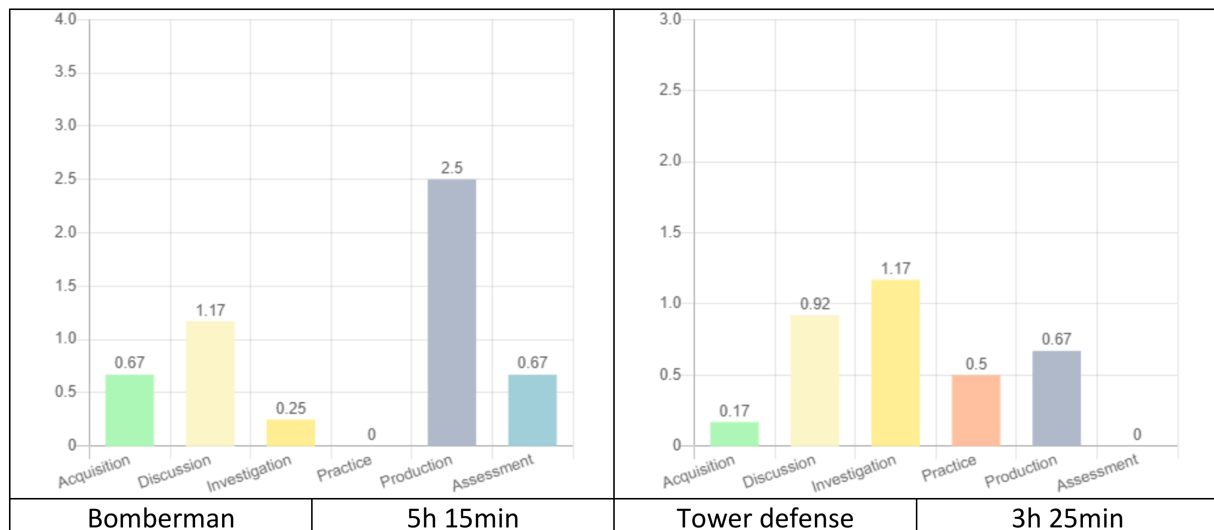
Tato kapitola zahrnuje neúplné a úplné větvení. Uvádějí se základy vnímání světa (třída **World**) hercem (třída **Actor**). Studenti budou schopni psát kód s využitím podmínek.

Stav projektu v této kapitole otevírá učiteli možnosti zadávat úlohy jako "použij instance třídy **Direction** a **Orb** na navigaci nepřítele (**Enemy**) tak, aby se pohyboval v požadovaném směru", s omezujícími podmínkami jako použití maximálního počtu instancí dané třídy nebo úlohy typu "předpovídej pohyb v požadovaném nastavení světa".

V tabulce 6 je shrnuté porovnaní pracovní zátěže kapitoly větvení mezi projekty Bomberman a Tower defense. Mezi projekty je zřetelný rozdíl. Tower defense rozděluje TLA typu tvorba na posílení zkoumání a procvičování. To umožňuje víc experimentovat a nechává otevřené dveře pro kreativitu studentů. Všimněte si nízký počet akvizičních TLA. Je to proto, že

a) není je zavedeno vícenásobné větvení
b) je kladen důraz na TLA typu zkoumání.

Tabulka 6: Porovnaní pracovní zátěže kapitoly větvení mezi projekty Bomberman a Tower defense



3.1. *Sledování stavu nepřítele*

Vytvořte instanci třídy **Enemy** a umístěte ji do středu hrací plochy. Otevřete okno s vnitřním stavem instance a umístěte ho tak, aby bylo viditelné po celou dobu běhu aplikace. Potom spusťte aplikaci a pozorujte, jak se mění hodnoty atributů **x**, **y** a **rotation** v instanci třídy **Enemy**. Jakým způsobem se tyto hodnoty mění při pohybu (nahoru, dolu, vlevo a vpravo) a při otáčení?

3.2. *Přidání detekce hraníce světa*

Do těla metody **act()** přidejte kód na otočeni nepřítele o 180° po dosáhnutí okraje světa.

Commit: [4927c3ff7eb39b51ba2738f2ab500fd6c32e3bb4]

3.3. Přidání tříd Direction a Orb

Vytvořte dvě nové třídy jako potomky třídy **Actor**. První z nich bude třída **Direction** a druhá z nich bude třída **Orb**. V grafickém editoru připravte vhodné obrázky (maximálně 50x50 bodů). Potom tyto obrázky přiřaďte k příslušným nově vytvořeným třídám.

Commit: `4ed6b37e6d481181d8b340639aa03391406b6c2e`

3.4. Přidání detekce kolizí

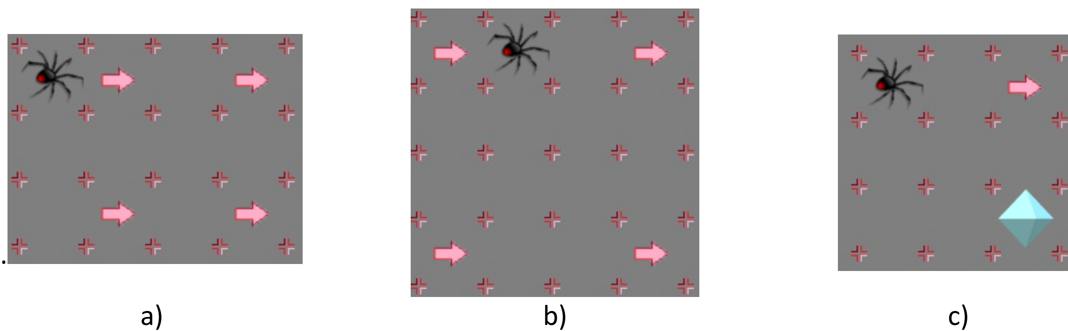
Do metody **act()** třídy **Enemy** přidejte kód, který zabezpečí, že:

- nepřítel se otočí o 90° v směru hodinových ručiček, pokud vstoupí do buňky, která obsahuje instanci třídy **Direction**,
- nepřítel se otočí o 90° proti směru hodinových ručiček, pokud vstoupí do buňky, která obsahuje instanci třídy **Orb**.

Commit: `968e6f195e3def25e11bc41b664ba1715f7da11d`

3.5. Předvídání pohybu nepřítele na vlastním uspořádaní

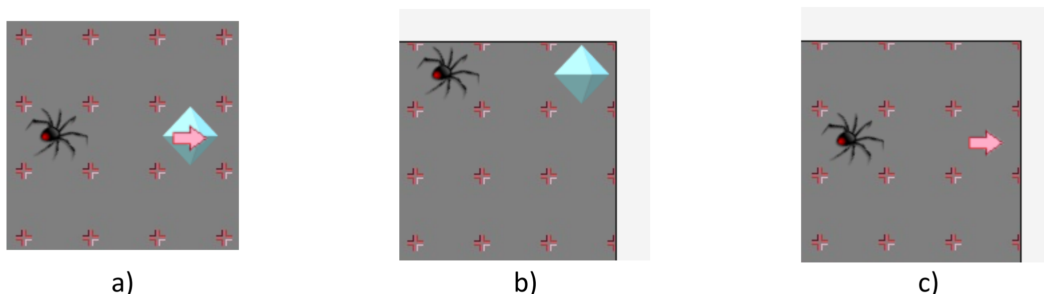
Připravte si různé konfigurace pro pohyb, přičemž inspiraci najdete na obrázcích níže. Odhadněte, jakým způsobem se bude pohybovat nepřítel. Spusťte aplikaci. Odpovídá Vaše predikce tomu, co pozorujete? Co způsobilo rozdíly mezi predikcí a skutečností?



Obrázek 5: Konfigurace vlastních nastavení instancí na předpovídání pohybu instance třídy Enemy

3.6. Předvídaní pohybu nepřítele při specifickém uspořádání

Připravte si situaci tak, jak je znázorněná na obrázku níže. Odhadněte, jakým způsobem se bude pohybovat nepřítel. Spusťte aplikaci. Odpovídá Vaše predikce tomu, co pozorujete? Co způsobilo rozdíly mezi predikcí a skutečností?



Obrázek 6: Konfigurace specifické nastavení instancí na předpovídání pohybu instance třídy Enemy

3.7. Použití úplného větvení při detekci kolizí

Z minulé kapitoly je vidět, že problém nastává, pokud je instance třídy **Orb** nebo **Direction** na okraji světa, popřípadě jsou obě instance ve stejné buňce. Při neúplném větvení dochází ke kolizím, resp. k opakovanému otáčení. V podstatě je splněno více podmínek najednou. Proto je potřeba změnit kód metody **act()** třídy **Enemy** tak, aby došlo jen k jednomu otočení. Je nutné použít úplné větvení. Vytvořte

kaskádu podmínek. Nejdůležitější kontrolou (tedy první) je detekce hrany. Druhá nejdůležitější je kontrola dotyku instance třídy **Direction.** Poslední je kontrola dotyku instance třídy **Orb**. Upravte dle těchto pravidel metodu **act()**.

Commit: f017de8b49d4fc77f62afac4d842429560bcfb8b

3.8. Předvídání pohybu nepřítele na základě předchozích nastavení

Znovu projděte úlohy 3.5 a 3.6. Co se změnilo?

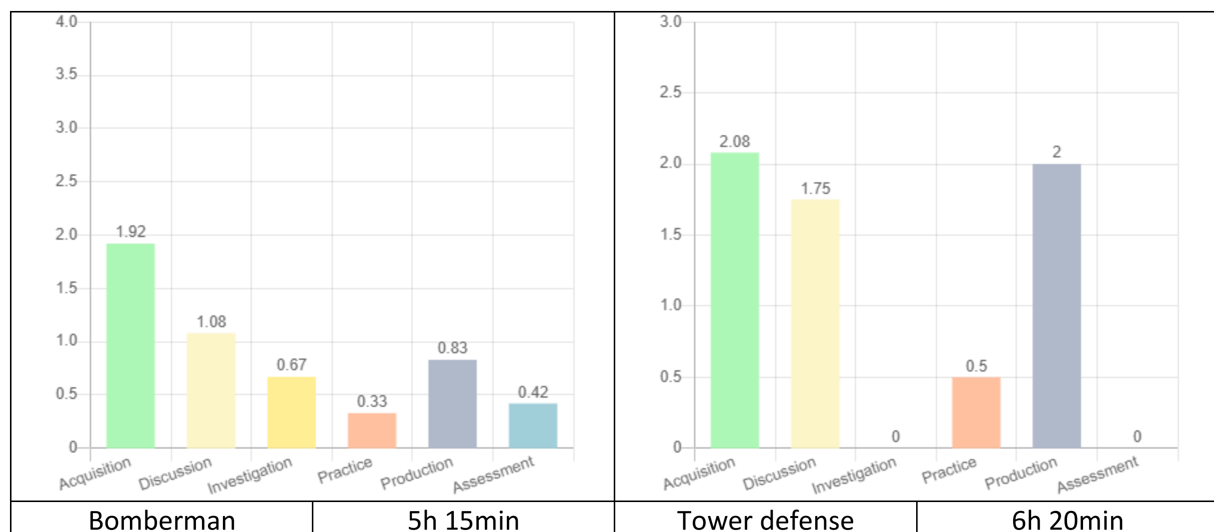
4. Proměnné a výrazy

Tato kapitola se věnuje proměnným a výrazům.

Stav projektu v této kapitole otvírá podobné možnosti jako v předcházející kapitole. S trochou kreativity je možné přidat další třídy jako je třída **Direction**, které způsobí různé chování při zásahu instancí třídy **Enemy** (např. teleporty, tunely atd.). O těchto třídách se dá se studenty diskutovat a příslušná implementace se může zadat jako domácí úkol.

V tabulce 7 je shrnuto porovnání pracovní zátěže kapitoly proměnné a výrazy mezi projekty Bomberman a Tower defense. Podívejte se na podobnosti návrhu předcházejících a aktuálních kapitol mezi oběma projekty. Tato kapitola je více orientována na tvorbu. Naopak méně na zkoumání oproti projektu Bomberman.

Tabulka 7: Porovnaní pracovní zátěže kapitoly proměnné a výrazy mezi projekty Bomberman a Tower defense



Bomberman	5h 15min	Tower defense	6h 20min

4.1. Otáčení ve směru

Změňte kód v metodě **act()** třídy **Enemy** tak, aby se její instance otáčela stejným směrem jak určuje instance třídy **Direction** (budou mít stejnou rotaci). Použijte metodu **getOneIntersectingObject(_cls_)** na uložení instance do správné lokální proměnné (**Direction direction** - budete muset použít přetypování, protože návratová hodnota je typu **Actor**; před voláním metody **getOneIntersectingObject** napište **(Direction)**, k přetypování se dostaneme později). V okamžiku, kdy byla získána nějaká instance příslušné třídy, metodou **getDirection()** získáte její natočení pro nepřítele (pokud chcete, uložte ho) a potom ho nastavte pro nepřítele (**this**) pomocí metody **setDirection(int)**. Otestujte své řešení.

Commit: 97dddc4beba40ac785c7413bb245ba849cd956d2

4.2. Přejmenování třídy MyWorld na Arena

Dejte třídě **MyWorld** lepši jméno. Přejmenujte ji na **Arena**. Nezapomeňte na příslušné přejmenování konstruktoru.

Commit: `aaf73c9bfd9f76a2a1e504f5e78d2976f1cada12`

4.3. Vytvoření uspořádání arény

Vytvořte vlastní rozložení instanci ve třídě **Arena**. Vyplňte konstruktor třídy. Přidejte jednu instanci třídy **Enemy**, jednu instanci třídy **Orb** a aspoň jednu instanci třídy **Direction**. Pokud chcete přidat instanci potomka třídy **Actor**, můžete použít následující šablonu:

1. deklarujte a inicializujte proměnnou požadovaného typu (potomek třídy **Actor**),
   Enemy e = new Enemy();
2. nastavte vlastnosti pomocí vhodných metod,
   e.setRotation(90);
3. vložte ji do světa (**Arena**) pomocí metody **addObject(Actor)**.
   this.addObject(e, 6, 0);

Otestujte své řešení.

Commit: `8b105ea2eaf697f08c321efe687ddd31e2d0a041`

4.4. Identifikace problému s pohybem a návrh řešení

Zjistěte, co způsobuje problémy s pohybem. Jakým způsobem se dají tyto problémy vyřešit?

***Nepřítel** se v současnosti pohybuje v jednom kroku 2 buňkami, což způsobuje problémy s pohybem. Rychlost nepřítele můžeme upravovat jinak. Instance třídy **Enemy** se bude vždy pohybovat jen 1 buňkou v jednom kroku. Zavedeme však zpoždění pohybu – instance třídy **Enemy** se bude pohybovat až po daném počtu (**delay**) zavolaní metody **act()**.*

4.5. Atribut moveDelay třídy Enemy

Přidejte do třídy **Enemy** nový atribut typu **int** s názvem **moveDelay**. Vytvořte parametrický konstruktor s parametrem na inicializaci tohoto atributu. Inicializujte atribut pomocí parametru. Podle těchto změn upravte kód ve třídě **Arena**.

Commit: `6092489ce57541e77ae4e2ee886b20853df9f8a4`

4.6. Pohyb nepřítele s dodržením zpoždění pohybu

Aktualizujte metodu **act()** třídy **Enemy** tak, aby se pohybovala po **moveDelay**-tom volaní metody **act()**. Zaveďte nový atribut **nextMoveCounter**. Inicializujte ho na **0**. Upravte metodu **act()** tak, aby volala **this.move(1)** jen v okamžiku, kdy **nextMoveCounter** dosáhne hodnotu **0**. Po vykonaní pohybu nastavte **nextMoveCounter** na hodnotu **moveDelay**. Pokud se instance třídy **Enemy** nemohla pohnout (protože **nextMoveCounter** nedosáhlo **0**), snižte **nextMoveCounter** o **1**.

Commit: `bf26e6ed23911ccb712fae3e243cdedff3a89a7f`

4.7. Parametrický konstruktor třídy Direction

Přidejte parametrický konstruktor do třídy **Direction** s jediným parametrem **rotation** typu **int**. Vytvořenou instanci v těle konstruktoru otočte podle parametru. Podle uvedených změn upravte kód ve třídě **Arena**.

Commit: `3c4b9ef57ab17bac2a0abc7fc5e76ea4b6e27e4b`

4.8. *Přetižte konstruktory ve třídě Direction*

Přetižte konstruktory ve třídě **Direction** přidáním neparametrického konstruktoru. V těle neparametrického konstruktoru zavolejte parametrický konstruktor s parametrem **direction** rovným **0**. Podle toho upravte kód ve třídě **Arena** – kde je to možné, volejte neparametrickou verzi konstruktoru třídy **Direction**.

Commit: 1e67e67523c66acea4e93363c9a3173302f424c8

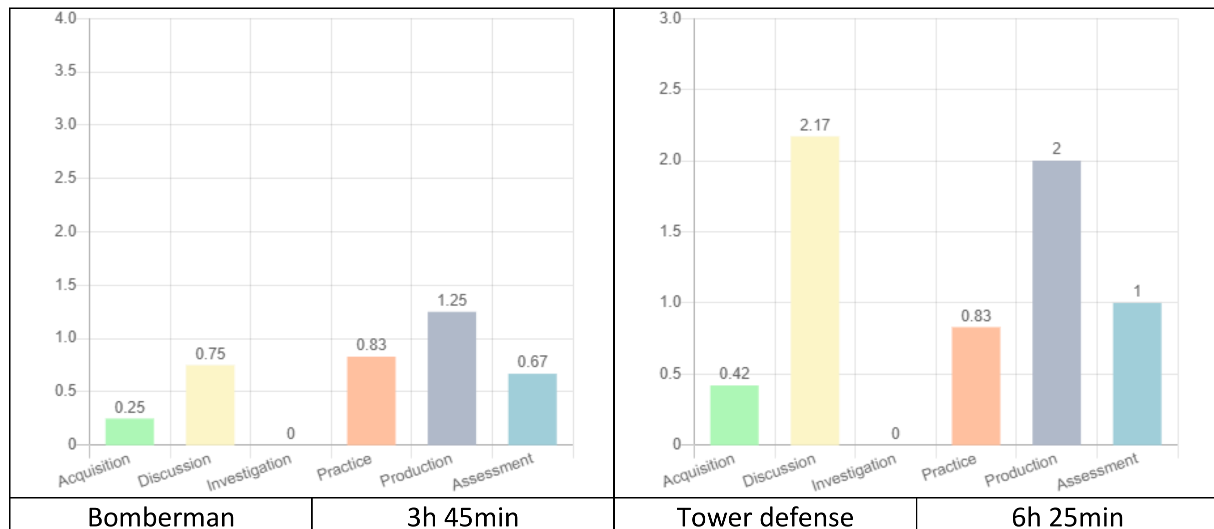
5. Asociace

Nejdůležitější kapitola tohoto projektu je zaměřená na asociaci. V rámci diskuse se zjišťuje, jak může spolupráce různých objektů přinést složité chování, přestože jsou kódy v objektech lehce pochopitelné a udržovatelné. Na znázorněni spolupráce objektů a rozdělení algoritmu mezi spolupracujícími objekty se používají sekvenční diagramy UML. Tento diagram se dá sestavit v době diskuse se třídou.

Projekt je možno prohlásit za dokončený po této kapitole. Následující kapitoly představují větší variabilitu aplikace se zaměřením na výhody OOP při jeho správném použití.

V tabulce 8 je shrnuté porovnaní pracovní zátěže kapitoly asociace mezi projekty Bomberman a Tower defense. Pochopení asociace považujeme za nejdůležitější kompetenci při využití tohoto projektu na výuku OOP. Proto jsme výrazně posilnili TLA typu tvorba a diskuse. Všimnete si, že posílená je i TLA typu hodnocení. TLA tohoto typu jsou navrhnuty tak, aby se využili dříve vykonané TLA v trochu jiném kontextu.

Tabulka 8: Porovnaní pracovní zátěže kapitoly asociace mezi projekty Bomberman a Tower defense

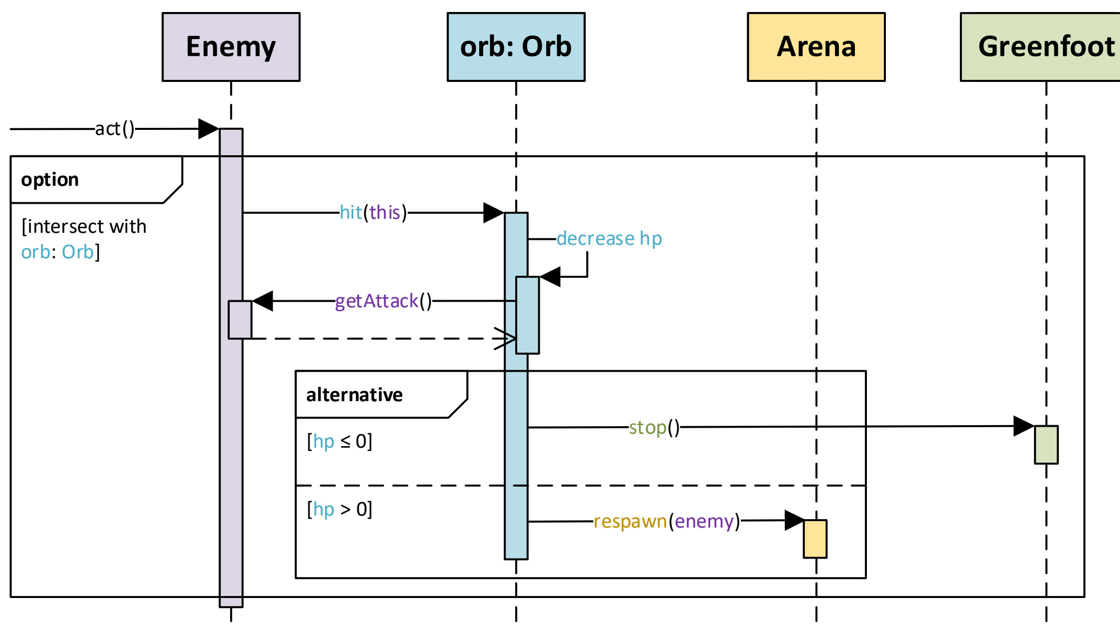


5.1. *Diskuse o tom, co by se mělo stát, když nepřítel dosáhne cílové místo*

Když se nepřítel dostane k cílovému místu, toto místo si sníží HP. Pokud je HP = 0, hra se ukončí, jinak se nepřítel opět vygeneruje v aréně.

5.2. *Diskuse o tom, jak by se měla instance třídy Enemy interagovat s příslušnými objekty pomocí zpráv při zásahu instance třídy Orb*

Algoritmus se rozloží mezi spolupracující objekty.



Obrázek 7: UML sekvenční diagram interakce instance třídy Enemy s jinými objekty při dosáhnutí instance třídy Orb

5.3. Atributy attack třídy Enemy a hp třídy Orb

Přidejte nový atribut typu **int** s názvem **attack** do třídy **Enemy**. Přidejte parametr do konstruktoru na inicializaci tohoto atributu. Inicializujte atribut pomocí parametru.

Přidejte nový atribut typu **int** s názvem **hp** do třídy **Orb**. Přidejte parametrický konstruktor s parametrem na inicializaci tohoto atributu. Inicializujte atribut pomocí parametru.

Upravte kód v třídě **Arena**.

Commit: 4ca1e9f25685990d2bdfe5b610c28422e0944f95

5.4. Získaní hodnoty atributu attack třídy Enemy

Vytvořte getter - metodu na získaní hodnoty privátního atributu **attack** ve třídě **Enemy**.

Commit: 72b7456ea4cc11416c57d72c89b6a7f7e9266e3e

5.5. Vytvoření a testovaní metody respawn(Enemy) třídy Arena

Přidejte metodu **respawn** bez návratové hodnoty s jedním parametrem typu **Enemy** do třídy **Arena**. V metodě nastavte polohu a rotaci nepřítele na stejné hodnoty jako při vytvoření v konstruktoru.

Otestujte metodu. Po vytvoření instance třídy **Arena** nespouštějte aplikaci. Namísto toho přesuňte myší instanci třídy **Enemy**. Potom vyvolejte kontextové menu instance třídy **Arena** (je třeba kliknout pravým tlačítkem do arény na místo, kde není žádná instance jiné třídy) a vyberte položku **respawn**. Pokud chcete vyplnit parametr, ujistěte se, že aplikace je pozastavená a soubor s parametrem je aktivní (s blikajícím kurzorem vevnitř). Pokud ano, klikněte levým tlačítkem myši na instanci třídy **Enemy**. Sledujte, jaký výraz se vytvořil v okně. Potom klikněte na tlačítko OK a sledujte, co se stane.

Commit: 43a221876b8acb4fd507175ec4c8f520121d1ab1

5.6. Vytvoření a testovaní metody hit(Enemy) třídy Orb

Přidejte metodu **hit** bez návratové hodnoty s jediným parametrem typu **Enemy** do třídy **Orb**. Tělo metody ponechte prázdné.

Otestujte volaní metody. Použijte podobné kroky jako výše, avšak vyvolejte kontextové menu instance třídy **Orb**. Sledujte, jaký výraz se vytvořil v okně.

Commit: fe03d520260f172066be35055a901487bf7c2ff7

5.7. Zavolání metody hit(Enemy) třídy Orb z třídy Enemy

Změňte kód v metodě **act()** třídy **Enemy** tak, aby se metoda **hit()** zavolala v okamžiku, když instance třídy **Enemy** zasáhne instanci třídy **Orb**.

Odstraňte staré kódy, které způsobovaly otáčení nepřítele při zásahu instance třídy **Orb** a které způsobovaly, že se nepřítel odrážel od okraje světa.

Commit: 63f9c96717d9d2587b60095e3b249b0158c8587b

5.8. Implementace metody hit(Enemy) třídy Orb

Implementujte tělo metody **Orb.hit(Enemy)** s ohledem na analýzu vykonanou v úloze 5.2. Otestujte svoji aplikaci.

Commit: 84bcd7c128faaa9313b507f7438f826ae2f47d2c

5.9. Přidáni tříd Bullet a Tower

Přidejte třídy **Bullet** a **Tower**. Použijte podobné principy jako v úloze 3.3.

Commit: ece4df70042c8f60098e14ad2cee55514897d825

5.10. Diskuse o tom, jak se má pohybovat instance třídy Bullet a co se má stát, když dosáhne instanci třídy Enemy nebo okraj arény.

Kulka (Bullet) by se měla pohybovat, dokud nedosáhne nepřítele nebo okraj světa. Kulka nemění směr pohybu. Rychlost kulky je možné řídit pomocí podobného mechanismu jako v úloze 4.6

5.11. Implementace pohybu instance třídy Bullet

Použijte vědomosti získané v úlohách 3.2 3.4 a 4.6. Přidejte do kódu, kde by mělo dojít k interakci s instancí třídy **Enemy** dokumentační komentáře.

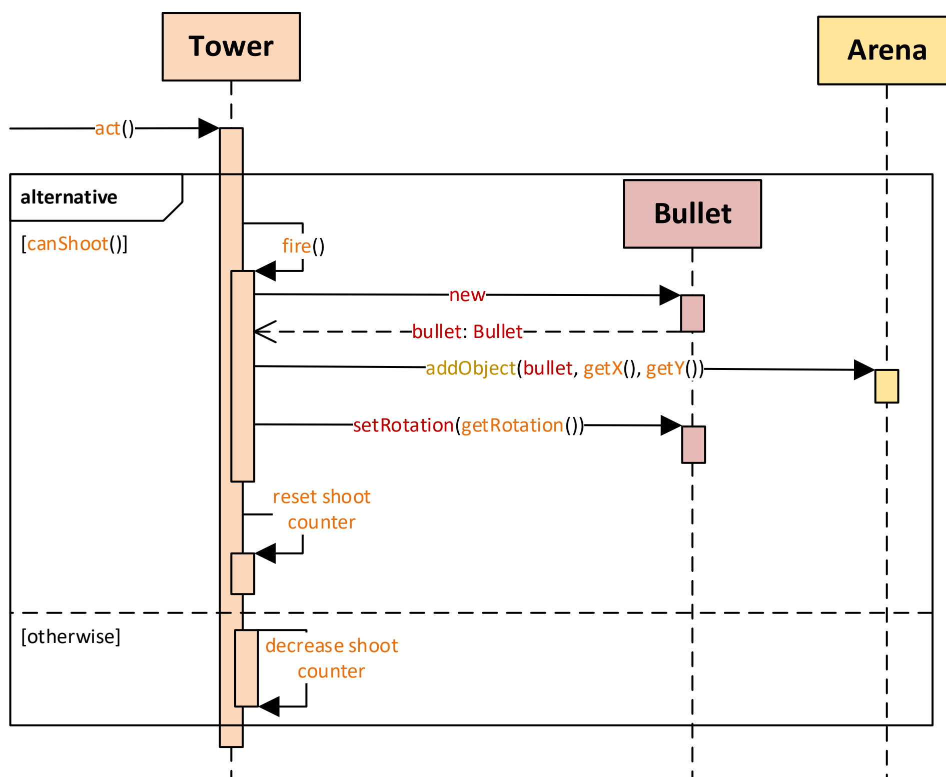
Commit: d372827a831381b2254f838041fa4d9a42e53b82

5.12. Diskuse o tom, jak instance třídy Tower vystřelí instanci třídy Bullet

*Nezapomeňte, že instance třídy **Tower** by neměla střílet při každém volaní metody **act()**. Inspirujte se mechanizmem použitým v části 4.6. Rozdělte příslušné kroky do metod třídy **Tower**.*

5.13. Diskuse o tom, jak by měla instance třídy Tower komunikovat s příslušnými objekty pomocí zpráv při střelbě

Použijte podobné principy jako v úloze 5.2.



Obrázek 8: UML sekvenční diagram interakce instance třídy Tower s jinými objekty při vytváření instancí třídy Bullet

5.14. Implementace střelby instance třídy Tower

Postupujte podle výstupu úlohy 5.13:

- nejprve připravte potřebné atributy a konstruktor,
- potom vytvořte metodu typu **boolean canShoot()** a metodu bez návratové hodnoty **void fire()** ve třídě **Tower** (první metoda zatím vrátí hodnotu **false**, druhá metoda zůstane prázdná) abyste je mohli použít v metodě **act()**,
- nakonec implementujte tělo metody **act()**.

Implementujte metodu **canShoot()** tak, aby vrátila **true** pokud počítadlo střelby dosáhne hodnotu **0**.

Implementujte metodu **fire()** následovně:

- zavolejte konstruktor třídy **Bullet** a vytvořenu instanci uložte do lokální proměnné (**Bullet bullet**),
- přidejte vytvořenou kulku do arény na stejné souřadnice jaké má instance třídy **Tower**,
- nastavte kulce stejný směr jako má instance třídy **Tower**.

Otestujte svoje řešení.

Commit: 62aec085954beacf996865a55bed312a09c675f2

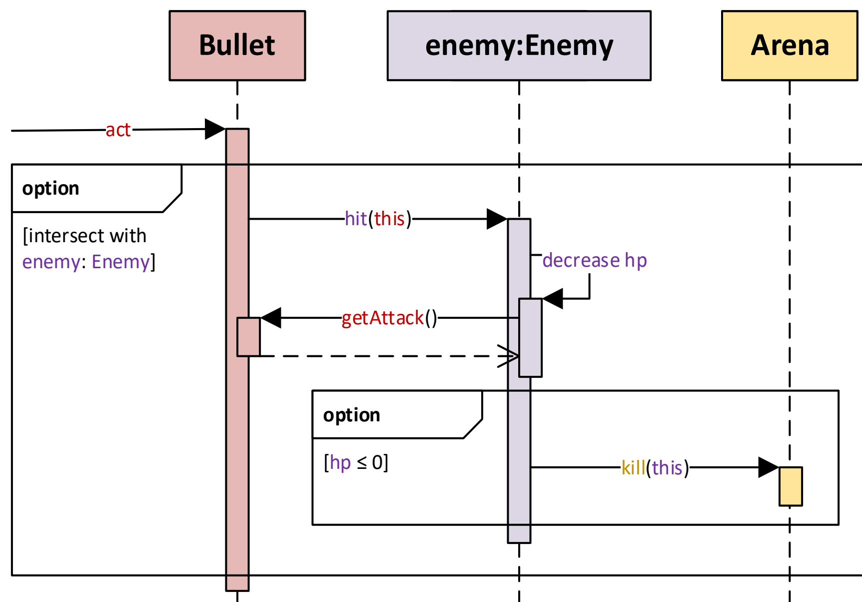
5.15. Věže v aréně

Proveďte přetížení konstruktoru třídy **Tower** tak, aby obsahoval i parametr **int rotation** (analogicky jako v úloze 4.8). Aktualizujte konstruktor třídy **Arena** tak, aby se instance třídy **Tower** umístily podle potřeby. Použijte správný konstruktor třídy **Tower**.

Commit: bfb6a271f490c341c760e654b3f86a87111c54cb

5.16. Diskuse o tom, jak by měla instance třídy Bullet komunikovat s příslušnými objekty pomocí zpráv

Použijte podobné principy jako v úloze 5.2.



Obrázek 9: UML sekvenční diagram interakce instance třídy Bullet s ostatními objekty při zásahu instance třídy Enemy

5.17. Implementace zásahu instance třídy Enemy instancí třídy Bullet

Postupujte podle výstupu úlohy 5.16:

- nejprve připravte atributy a metody (analogicky podle úloh 5.3, 5.4, 5.5 a 5.6),
- potom zavolejte metodu **hit(Bullet)** třídy **Enemy** z instance třídy **Bullet** (analogicky jako v úloze 5.7) kde byl zanechaný komentář v úloze 5.11,
- na závěr implementujte metodu **hit(Bullet)** ve třídě **Enemy** (analogicky jako v úloze 5.8).

Otestujte svoje řešení.

Commit: `dcfe31bc006b7f3dcd8b8b759cc1be901c32913c`

5.18. Vznik nepřátel a konec hry

Na vyvolaní vytvářeni nepřátel použijte metodu **act()** třídy **Arena**. Správně implementujte pauzu mezi vytvořením nepřátel. Proces vytváření (vytvořit instanci třídy **Enemy**, přiradit jí vlastnosti, přidat jí do arény) implementujte v metodě **spawn()** třídy **Arena**. Počítaní vytvořených instancí třídy **Enemy** implementujte v atributu třídy **Arena** (inicializované na **0**, zvýšení při vytvoření, snížení při zabití). Změňte metodu **kill(Enemy)** ve třídě **Arena** – v okamžiku kdy je poslední nepřítel zabitý, hráč vyhrál hru – zastavte Greenfoot a vypište příslušnou správu na obrazovku.

Commit: `d48341a095561500af6032d5c8f56e201060f9a4`

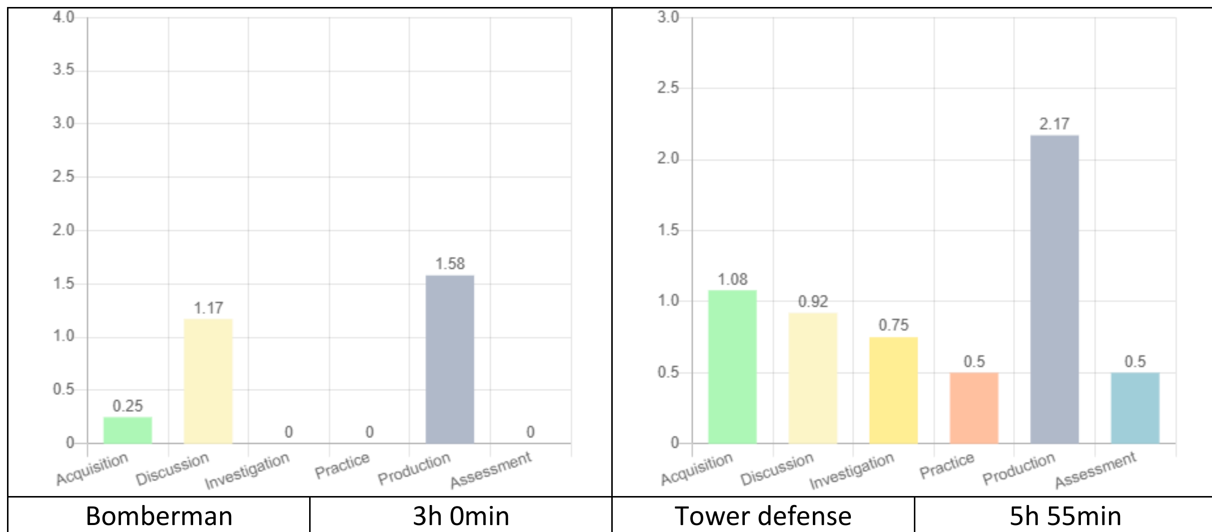
6. Dědičnost

Tato kapitola zavádí do projektu různorodost prostřednictvím dědičnosti. Zavedeme Liskové principy zastoupení (angl. Liskov substitution principle), abychom ukázali výhody OOP. Důrazně doporučujeme nechat studenty experimentovat a vymýšlet je vlastní potomky nepřátel a arén. Přestože budou potomci mít společné rozhraní, bude jednoduché vše spojit dohromady. Podobně jako v kapitole 4 může být vytvořeno velké množství domácích úloh.

Jak již bylo zmíněno, projekt se může nacházet v dokončené fázi i po předchozí kapitole. Proto v případě potřeby může vyučující toto kapitolu upravit tak, aby ukázal výhody dědičnosti a s ní spojenou univerzálnost jen na navrhovaných hierarchiích tříd **Arena** nebo **Enemy**. To povede ke zkrácení počtu hodin spojených s touto kapitolou.

V tabulce 9 je shrnuté porovnaní pracovní zátěže kapitoly dědičnost mezi projekty Bomberman a Tower defense. Návrh při experimentovaní se promítá do většího počtu TLA typu zkoumání, procvičování a tvorba. V této kapitole je zahrnutá rozsáhlejší teoretická část se zaměřením na Liskové principy zastoupení

Tabulka 9: Porovnaní pracovní zátěže kapitoly dědičnost mezi projekty Bomberman a Tower defense



Bomberman	3h 0min	Tower defense	5h 55min

6.1. Identifikace společných vlastností tříd Orb a Direction

*Instance tříd **Orb** a **Direction** nepotřebují specifickou implementaci v metodě **act()**. Reagují jen na zprávy. Můžeme proto zavést společného předka, díky kterému bude metoda **act()** prázdná a potomkové budou přehlednější.*

6.2. Přidáni třídy PassiveActor jako předka tříd Orb a Direction

Vytvořte novou třídu **PassiveActor**. Změňte kódy tříd **Orb** a **Direction** tak, aby byly potomky třídy **PassiveActor**. Odstraňte metodu **act()** z tříd **Orb** a **Direction** – je totiž zděděná ze třídy **PassiveActor**.

Commit: afe617814c07a5d885ed06479bf71deda8725f19

6.3. Zavedení abstraktnosti pro třídu PassiveActor

Při tvorbě metod třídy **PassiveActor** narazíme na metody, které nelze ve společné třídě implementovat a implementaci tedy musíme nechat právě na potomky. Jako příklad uveďme třídu **Obrazec** s potomky **Obdélník** a **Trojúhelník**. Každý obrazec bude mít implementované metody obvod a obsah, ale nelze je implementovat ve společné třídě. Pokud metodu třídy označíme jako abstraktní, tak tím v podstatě řekneme, že ji bude implementovat potomek. Třída obsahující abstraktní metodu musí být abstraktní. Proto přidáme k hlavičce třídy slovo **abstract**.

Commit: f7a5702cae29bf21c9c88620d01ef64e4127c21c

6.4. Identifikace společných vlastností tříd Bullet a Enemy

*Instance tříd **Bullet** a **Enemy** se v době svého života chovají podobně. Pohybují se stejným způsobem a také poté reagují na okolí. Můžeme zavést společného předka, který bude implementovat metodu **act()**, aby se pohybovaly stejným způsobem. Potomci se poté zaměří na svůj specifický účel.*

6.5. Přidání abstraktní třídy MovingActor jako předka tříd Bullet a Enemy

Použijte podobný postup jako v úloze 6.2.

Commit: 43e53b533563ce0a860b294ad9009f77409c48d4

6.6. Identifikace atributů tříd Bullet a Enemy potřebných na pohyb

Přezkoumejte metody **act()** příslušných tříd. Identifikujte atributy **moveDelay** a **nextMoveCounter**. Všimněte si, že kód metody **act()** odpovědný za pohyb je totožný.

6.7. Přesun kódu odpovědného za pohyb do třídy MovingActor

- Přesuňte atributy identifikované v části 6.6 z potomků **Bullet** a **Enemy** do třídy **MovingActor** (odstraňte je z potomků).
- Přidejte parametrický konstruktor do třídy **MovingActor** na inicializaci těchto atributů.
- Zavolejte konstruktor předka se správnými parametry z potomků **Bullet** a **Enemy**.
- Přesuňte kód odpovědný za pohyb v metodě **act()** z potomků **Bullet** a **Enemy** do předka **MovingActor** (odstraňte kód z potomků, zbytek metody tam ponechte).
- Zavolejte pomocí **super** metodu předka **act()** jako prvý řádek v metodě **act()** v potomcích **Bullet** a **Enemy**.

Commit: ca1f010a63445c1847b74259a1c6cd4817121db3

6.8. Vytvoření vlastních nepřátel

- Přidejte potomka třídy **Enemy**, který bude představovat různé nepřátele (např. **Frog** a **Spider**). Ujistěte se, že obrázky nepřesahují velikost buňky.
- Přidejte do tříd bezparametrický konstruktor, který bude volat konstruktor předka (třídy **Enemy**) s parametry specifickými pro každý druh nepřítele.
- Odstraňte metodu **act()** v potomcích (nebo přidejte volaní **super()**).

Commit: b0ac1fbe793548a32f7700c292aed631918c8388

6.9. Vytvoření vlastních nepřátel

Aktualizujte metodu **spawn()** třídy **Arena**. Vytvořte instanci třídy **Frog** nebo **Spider** a uložte jo do proměnné typu **Enemy**. Na rozhodnutí, která instance se má vytvořit, použijte libovolný druh rozhodnutí (může být náhodné, může být přesně spočítané atd.). Dbejte na to, aby se v aplikaci nezměnily žádné jiné kódy.

Commit: 8cd4397f585ec957bbc18ca98e01823f434a13a6

6.10. Diskuse o hierarchii arén

Diskutujte a navrhněte hierarchii tříd **Arena**. Potomci třídy **Arena** budou odpovědní za vlastní rozložení – pozici instancí tříd **Orb** a **Direction** a velikost arény. Tyto úlohy se budou vykonávat v konstruktoru potomků. Co bude potřebné zadat do parametrů konstruktoru třídy předka (**Arena**)? Nezapomeňte, že vše ostatní (vytváření, znovu vytváření a zabíjení nepřátel) se bude provádět ve třídě **Arena**.

Měli byste identifikovat potřebu nastavit a uložit polohu a natočení při vytváření nepřátel. To se vykoná pomocí atributů a příslušných parametrů konstruktoru. Kromě toho by měl konstruktor akceptovat i rozměry plochy.

6.11. Vytvoření univerzální arény

Vytvořte atributy **int spawnPositionX** , **int spawnPositionY** a **int spawnRotation** a použijte je v metodách **spawn()** a **respawn(Enemy)**. Přidejte parametry do konstruktoru třídy **Arena** na jejich inicializaci.

Přidejte další dva parametry do konstruktoru třídy **Arena** – **int width** a **int height**. Tyto parametry předejte konstruktoru předka.

Všimněte si, že **Arena** nemůže být automaticky vytvořená v prostředí **Greenfoot**, protože potřebuje parametry pro konstruktor. Proto nastavte třídu **Arena** jako abstraktní.

Commit: e9844d7d9b5f19969618b469ebc907d0fe3c1357

6.12. Vytvoření třídy DemoArena

Přidejte potomka **DemoArena** třídy **Arena**. Zavolejte konstruktor předka třídy **DemoArena** s parametry, které zabezpečí vytvoření arény se stejnými rozměry a vytvoření nepřátel stejným způsobem jako dříve.

Přesuňte kód odpovědný za rozložení instancí tříd **Direction**, **Orb** a **Tower** z konstruktoru třídy **Arena** do konstruktoru třídy **DemoArena**.

Vytvořte instanci třídy **DemoArena** – z kontextového menu třídy **DemoArena** vyberte položku „***new DemoArena()***“.

Commit: 6a6569774b5735f453a56c7cb2cdbf19d228eae9

6.13. Vytvoření vlastních arén

Pomocí podobného přístupu jako v časti 6.12 vytvořte další zajímavé potomky třídy **Arena**. O svůj kód se můžete podělit s ostatními studenty ve Vaší skupině.

7. Zapouzdření, statické metody a atributy

Poslední kapitola je zaměřená na správné použití soukromých metod, statických metod a statických atributů. Použití metod tříd a proměnných lze nahradit (netřídními) atributy a metodami ve třídě **Arena** (v kontextu našeho projektu je přítomna jen jedna instance třídy **Arena**). To umožní implementovat úlohy z této kapitoly ale s využitím už známých konceptů.

Zapouzdření jsme již použili u projektu Bomberman. Zde se nově použijí statické atributy a statické metody. Doposud byly všechny atributy a metody vázané na instanci dané třídy. Představme si situaci, kdy bychom ve třídě **Bullet** chtěli počítat vystřelené kulky. Počet vystřelených kulek je totiž společný pro všechny instance třídy **Bullet** a nezávisí na konkrétní instanci. Takový atribut se nazývá statický a jelikož je společný pro třídu, přistupuje se k němu přes název třídy, tedy např. **Bullet.count**. Podobně můžeme vytvořit například metodu, která vrátí počet kulek. Opět bude společná pro všechny instance dané třídy. Bude tedy statická. Statické atributy a statické metody mají ve své deklaraci slovo **static**. Poznamenejme, že je lze použít i v případě, že neexistuje žádná instance dané třídy. Statické atributy se inicializují v samotné definici.

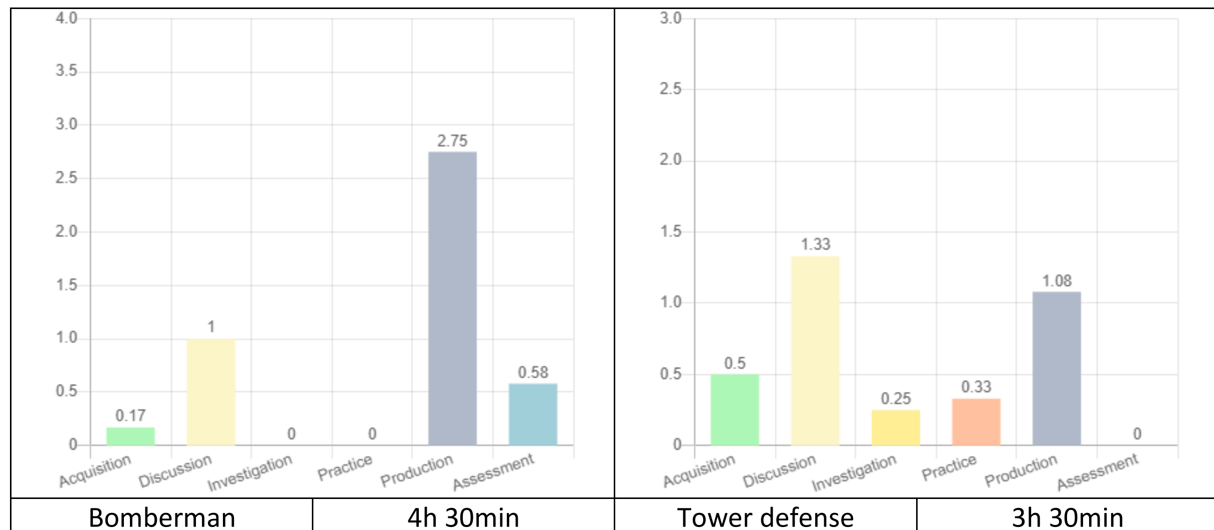
```
static int countOfBullets = 0;

static int numberOfBullets(){
…
```

V tabulce 10 je zahrnuté porovnání pracovní zátěže kapitoly zapouzdření mezi projekty Bomberman a Tower defense. Podobně jako v předcházejících kapitolách je tvorba rovnoměrněji rozdělená mezi

ostatní typy TLA. Vyšší množství akvizičních TLA pochází ze zavedení statických metod a proměnných. Jak bylo naznačeno, existuje možnost se těmto konceptům vyhnout, což by vedlo k redukci těchto TLA a co do výsledku k podobnému návrhu jako u projektu Bomberman.

Tabulka 10: Porovnání pracovní zátěže kapitoly zapouzdření mezi projekty Bomberman a Tower defense



Bomberman	4h 30min	Tower defense	3h 30min

7.1. Vytvoření třídy ManualTower

Vytvořte třídu **ManualTower** jako potomka třídy **Tower**. Připravte obrázky této třídy, kdy je pod kontrolou hráče a kdy ne. Přidejte dva konstruktory v podobném duchu jako konstruktory předka a zabezpečte volaní konstruktoru předka. Zařiďte aby metoda **act()** volala metodu **act()** z předka.

Přidejte instance této třídy do rozložení vybrané arény.

Commit: 63a02fa0c5080165cba8b467da08c4b65f31d0a8

7.2. Změna ovládání manuálně ovládané věže

Přidejte atribut **boolean isManuallyControlled** a inicializujte ho na **false**. Vytvořte metodu **void changeControl(boolean)** a změňte její atribut a příslušný aj obrázek.

Commit: 2257746b7dac5eaab7acc55d6493319230338f3a

7.3. Vyvolaní změny manuálního ovládání

Manuálně vyvolejte změnu způsobu ovládání vybrané instance třídy **ManualTower**. Sledujte změny vnitřního stavu podobně jako v části 1.6.

7.4. Řízení způsobu ovládání

Vytvořte soukromou metodu **void processUserControl()**. V ní se nejdříve zjistí, či bylo kliknuto na toto instanci. Pokud ano, změní se na ruční ovládání. Potom implementujte samotné ruční ovládání. Otestujte, či je instance v manuálním režimu, a pokud ano, získejte objekt **MouseInfo**. Po jeho získání, otočte instanci třídy **ManualTower** směrem k pozici kurzoru myši.

Před odevzdáním vykonání metody **act()** předka (**super.act()**) zavolejte metodu **processUserControl()** z metody **act()**. V metodě zkontrolujte, zda bylo na tuto instanci kliknuto. Pokud ano, zavolejte metodu **changeControl()** se správnou hodnotou.

Otestujte svoje řešeni opětovným vykonáním 7.3.

Commit: 6ec1f489576019a6493490f9e97797920b923869

7.5. Identifikace problému způsobu ovládání a návrh řešení

Identifikujte, co je problematické při manuálním ovládaní. Jak je možné tyto problémy vyřešit?

V současnosti není možné zrušit výběr věže. K aktuálně ovládané instanci by měla existovat evidence, která se deaktivuje, v okamžiku vybrání nějaké jiné instance. Přidejte evidenci manuálně ovládané věže.

7.6. Přidáni evidence manuálně ovládaných věží

Přidejte do třídy **ManualTower** atribut **controledInstance** typu **ManualTower**, který představuje odkaz na manuálně ovládanou instanci, a inicializujte ho na **null**. Tento atribut ale musí být společný pro všechny instance třídy **ManualTower**. Musí tedy být statický. Proto je použito slovo **static.** Zkontrolujte vnitřní stav třídy (z kontextového menu třídy **ManualTower** vyberte položku **Prohlížet**). Co bylo přidáno?

Commit: `c4739460bed583d2126de066acc6b1149d022990`

7.7. Změna manuálně ovládané věže z centralizovaného místa

Přidejte statickou metodu **changeControlledInstance()** třídy **ManualTower** na změnu manuálně ovládané věže (metoda musí být statická – pracuje totiž se statickým atributem). Parametrem metody by měl být odkaz na instanci třídy **ManualTower**, která bude nově manuálně ovládána.

Pokud se liší parametr metody od ručně ovládané instance (ve statickém atributu), použijte metodu **changeControll()** se správnými parametry. Nejprve pomocí **changeControll()** nastavte původní manuálně ovládanou instanci na **false**. Pak Změňte statický atribut třídy **ManualTower** manuálně ovládané instance na parametr nové metody (tj. nově manuálně ovládané věž). Na závěr nastavte pomocí metody **changeControll()** novou manuálně ovládanou instanci na **true**. Nezapomeňte zpracovat možné **null** odkazy. Mohla by totiž nastat situace, že manuálně ovládaná věž není žádná.

Otestujte svoje řešení. Z kontextového menu třídy **Tower** vyberte položku s nově vytvořenou metodou. Na vyplnění parametru můžete použít podobný princip jako v úloze 5.5.

*Měli byste si všimnout, že se nově vytvořená metoda třídy nevolá konzistentně, instance třídy **ManualTower** obchází evidenci při zpracovaní vstupu, což způsobuje problémy.*

Commit: `9dc6d8dd4dcbbd71edb8009c1a72403dea1a0ee0`

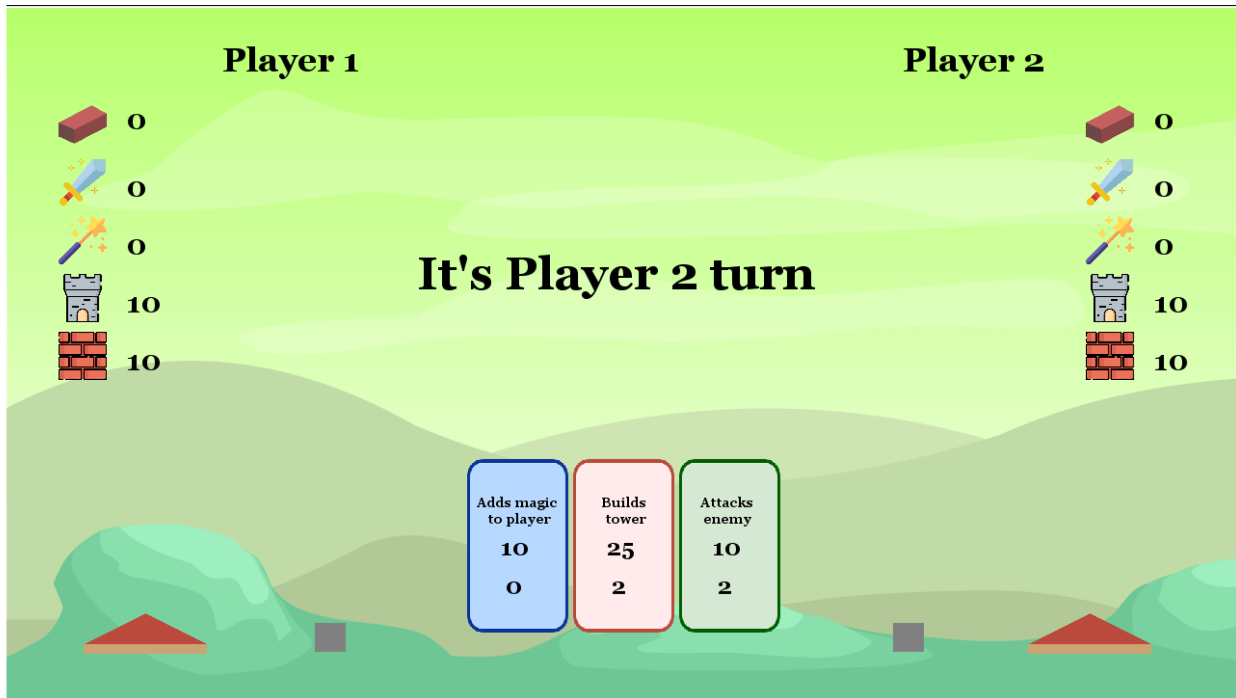
7.8. Vyvolaní změny manuálně ovládané věže

Vyvolejte **changeControlledInstance(ManualTower)** třídy **ManualTower** z příslušných míst. Nakonec změňte metodu **changeControl(boolean)** třídy **ManualTower** na soukromou. Pozorujte změny v rozhraní instance třídy **ManualTower** podobně jako v 1.6.

Commit: `c052bbb6aa4c7e690d4d8cf55d3831028fa2b9e3`

3.3. Projekt Mravenci

Mravenci (angl. Ants) je kartová hra pro dva hráče. Každý hráč má svoji vlastní věž, stěnu a zdroje, kterými jsou cihly, meče a kouzla. Jedno kolo hry se skládá z akce, při které hra poskytne hráči 3 náhodné karty a on si jednu vybere. Existují tři typy karet - karty staveb, bojové karty a kouzelné karty. Stavební karty lze použit na zvětšení vlastní věže nebo hradby, bojové karty na útok na nepřátelského hráče a magické karty na zvýšení počtu vlastních surovin nebo ukradení těch nepřátelských.



Zdrojové kódy jsou k dispozici na:
https://gitlab.kicon.fri.uniza.sk/oop4fun/project-ants

Návrh vzdělávaní je dostupný na:
http://learning-design.eu/en/preview/67aa1d089763d07f29809d42/details

3.3.1. Rozdělení projektu

Projekt mravenci je rozdělený do 6 kapitol.

1. Prostředí Greenfoot, definice třídy, základní práce s třídou 55
2. Zapouzdření, kompozice, metody 56
3. Konstruktory, složitější volání metod (práce s grafikou v prostředí Greenfoot) 58
4. Větvení, podmíněné vykonávání 59
5. Algoritmus, výčtový typ (enum), pole 60
6. Zpracování uživatelského vstupu, logika hry 63

Aplikovaná témata z light OOP jsou:

- třídy, objekty, instance,
- metody, předávání argumentů metod,
- konstruktory,
- atributy,
- statické proměnné a metody

- zapouzdření,

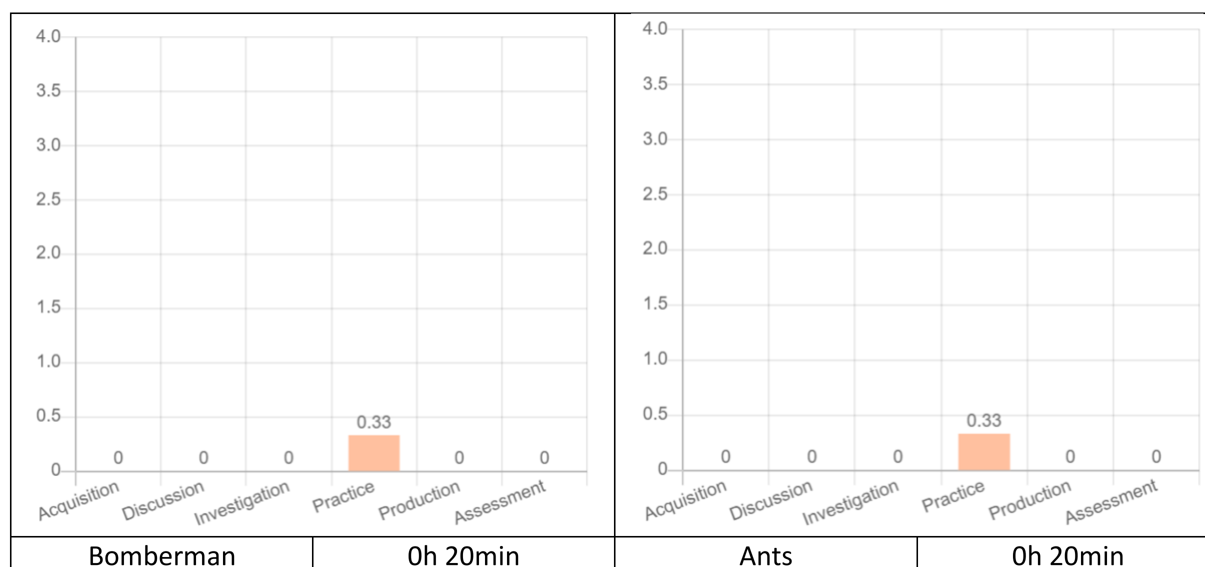
1. Prostředí Greenfoot, definice třídy, základní práce s třídou

Tato kapitola je věnovaná tvorbě projektů. Studenti budou schopni vytvořit nový projekt v prostředí Greenfoot, vytvořit třídu (jako potomka třídy **Actor**), vybrat obrázek pozadí, vytvořit instanci vytvořené třídy a poslat jí zprávu.

Vytvořte nový projekt. Dejte mu vhodný název (např. **Ants**) a uložte ho na vhodné místo.

V tabulce 11 je shrnuté porovnaní pracovní zátěže kapitoly úvod do prostředí Greenfoot mezi projekty Bomberman a Mravenci. Celková pracovní zátěž obou projektů je v této kapitole stejná.

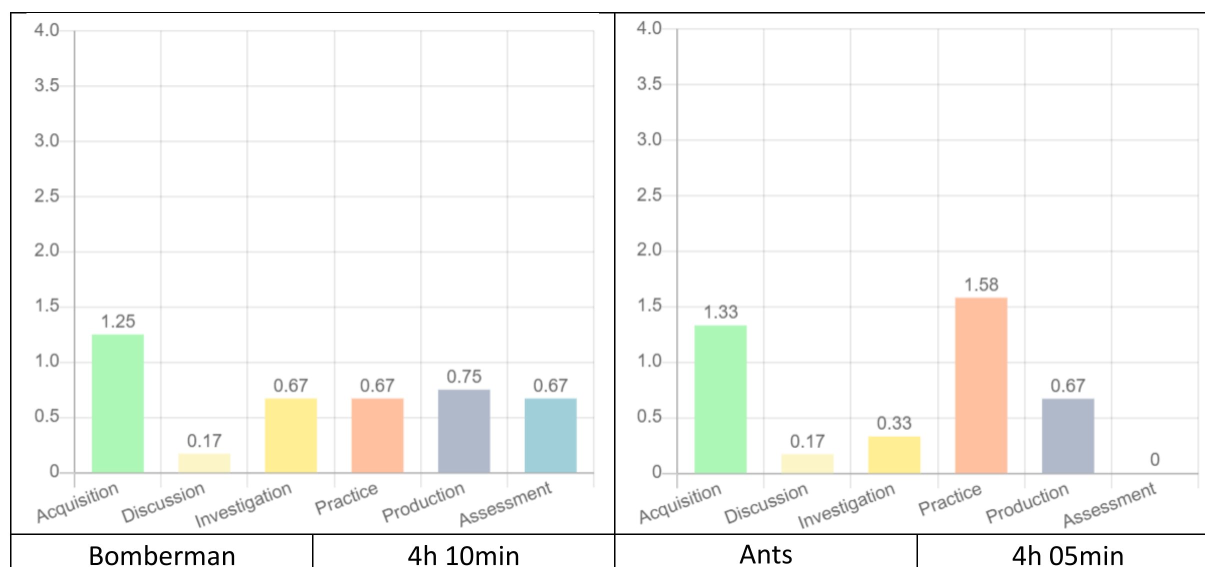
Tabulka 11: Porovnaní pracovní zátěže kapitoly úvod do prostředí Greenfoot mezi projekty Bomberman a Mravenci



Bomberman	0h 20min	Ants	0h 20min

V tabulce 12 je shrnuté porovnaní pracovní zátěže kapitoly definice třídy a základní práce se třídami mezi projekty Bomberman a Mravenci. Celková pracovní zátěž obou projektů je podobná. Hlavní rozdíl je v procvičování a hodnocení. Jak bude uvedené v další části, některé praktické úlohy mohou být zadané i jako hodnocení, což může projekt Mravenci ještě víc vyvážit.

Tabulka 12: Porovnaní pracovní zátěže kapitoly definice třídy a základní práce se třídami mezi projekty Bomberman a Mravenci



Bomberman	4h 10min	Ants	4h 05min

1.1. Úvod do prostředí Greenfoot

Vytvořte nový projekt v prostředí Greenfoot a seznamte studenty se základními prvky, uživatelským rozhraním atd. Počáteční stav repozitáře obsahuje také soubory, které lze v projektu použít.

Commit: *2f0658d99a7abcbad6399f63c369dcd4c053af2a*

1.2. Vytvoření třídy Wall

Vytvořte třídu **Wall** jako potomka třídy **Actor**. Představte studentům pojmy jako třídy, hierarchie tříd, instance tříd atd.

Commit: *cd163c8a3b1c952760a3a9e24ec6a322939a8ea6*

1.3. Vytvoření třídy Tower

Podobně jako v předchozí části vytvořte třídu **Tower**. Můžete to nechat na studentech jako samostatnou úlohu.

Commit: a0c1405183704f61156732c9bd55cdc921d95adc

1.4. Definovaní atributu/pole třídy

Představte studentům pojmy pole/atribut, primitivní typy atd. Pokuste se identifikovat pole ve vašich třídách (naveďte studenty konkrétně na výšku (**height**)) a definujte je v třídě **Wall**.

Commit: *b6bd179478e1440249080d4cdcc00b4428bef080*

1.5. Přiřazení hodnoty atributu/poli

Diskutujte o hodnotách a přiřazení polí. Přiřaďte k atributu **height** třídy **Wall** hodnotu **10**.

Commit: *917861df37d13c09d840008e0c7f7d263ea56c95*

1.6. Definovaní a přiřazení hodnoty atributu/pole pro třídu Tower

Jako samostatnou práci nechte studenty zopakovat tento postup pro třídu **Tower**.

1.7. Konstruktory tříd

Diskutujte o instancích, třídách a konstruktorech. Přesuňte přiřazení hodnoty do konstruktoru.

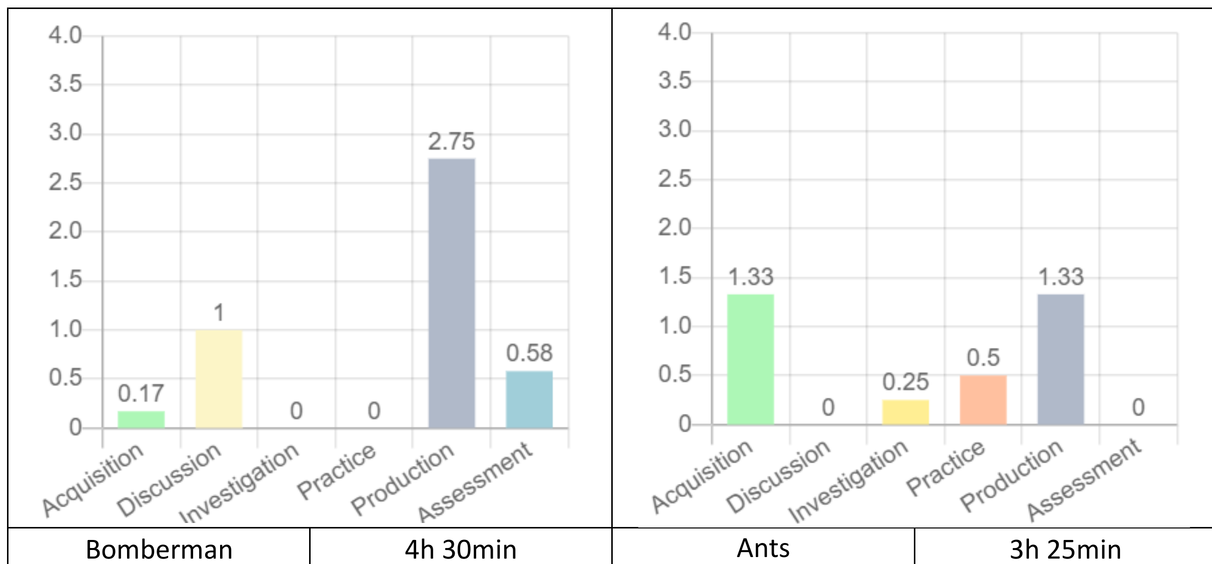
Commit: *a85930c312f079d65137eba8e147a8ba2b99e84f*

2. Zapouzdření, kompozice, metody

V této kapitole jsou uvedené základní principy OOP, konkrétně pojmy jako zapouzdření, kompozice a metody. Studenti se dozví, jak a proč by měly být pole/atributy zapouzdřené a neměly by být deklarované jako veřejné, jak se objekty skládají z jiných objektů a jak vytvářet metody a volat je. Studenti v této části vytvoří instanci třídy **Player** a předají ji pole typu objekt, konkrétně **Wall** a **Tower**.

Tabulka 13 ukazuje rozdíly ve dvou podobných kapitolách - zapouzdření z projektu Bomberman a zapouzdřeni, kompozice a metody v projektu Mravenci. Jak je vidět, projekt Mravenci se více zaměřuje na akvizici a na druhé straně projekt Bomberman se víc zaměřuje na tvorbu a diskusi. Je to způsobeno tím, že se tato kapitola v projektu Mravenci skládá z více oblastí jako zapouzdření. Proto je potřebné také více akvizice. Celková pracovní zátěž je v projektu Mravenci o hodinu menší než v projektu Bomberman, protože tyto pojmy jsou vysvětlené více povrchně, ale v dalších částech se tyto vědomosti zkonsolidují.

Tabulka 13: Porovnaní pracovní zátěže kapitoly zapouzdření, kompozice, metody v projektu Mravenci a podobné kapitoly v projektu Bomberman - zapouzdření



Bomberman	4h 30min	Ants	3h 25min

2.1. Definovaní metod

Pobavte se se studenty o zapouzdření a vysvětlete jim, proč není dobré např. vytvořit výšku (**height**) stěny jako veřejnou vlastnost a raději ji zapouzdřit v příslušné metodě (getter). Potom vytvořte metodu (getter) **getHeight()** ve třídě **Wall**.

Commit: *33af3b2d18a7b6c759c16b1bb0a4632f648a4d85*

2.2. Definovaní metod s parametry

Vysvětlete parametry metody a definujte metodu **increaseHeight()** ve třídě **Wall**, která zvýší výšku stěny o zadané číslo.

Commit: *50cfefc2747ee3150a16f66f439a9391fbff922a*

2.3. Zopakování tvorby metod pro třídu Tower

Jako samostatnou úlohu můžete studentům zadat úlohu, aby si předtím vysvětlené přístupy zopakovali i na třídě **Tower**. Táto úloha nemá související commit, protože práce je poměrně jednoduchá. V dalším commitu jsou již změny také pro tuto úlohu, takže pokud si nejste jisty výsledkem, můžete ho porovnat.

2.4. Kompozice objektů

Vysvětlete studentům, co je to kompozice a proč je potřebné mít pole s prvky typu instance třídy. Pokuste se identifikovat takové typy pro každého hráče v této hře. Potom vytvořte třídu **Player** a přidejte jí atribut typu **Wall** a **Tower**.

Commit: *50b6b825952c02f9743e08bd1bb8415aa6f08eef*

2.5. Vytvoření instance třídy

Vysvětlete konstruktor a vytvořte instanci tříd **Wall** a **Tower** v konstruktoru třídy **Player**.

Commit: *873174a2851f5b5e6054f8071412058fc97e6e2e*

2.6. Volaní metod instance

Vysvětlete studentům, jakým způsobem se mohou volat metody vytvořených instancí. Potom se je pokuste zapouzdřit tak, aby je bylo možné volat z prostředí mimo třídy **Player** prostřednictvím instance

této třídy. Vytvořte metody **getWallHeight()**, **getTowerHeight()**, **increaseWallHeight()** a **increaseTowerHeight()** ve třídě **Player**.

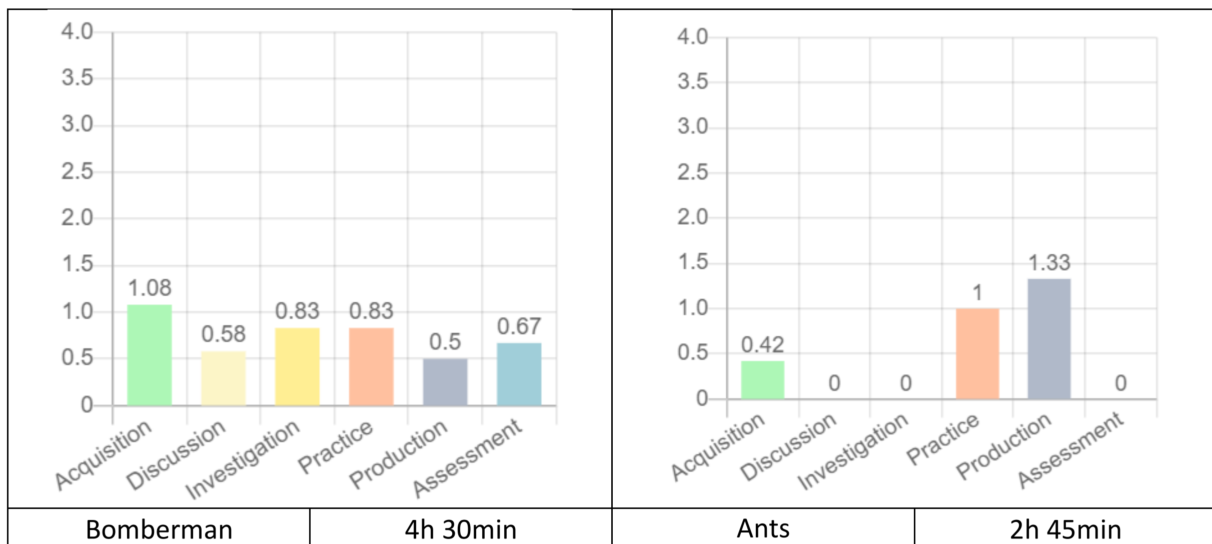
Commit: *b787d7f70ee6772b0185d5265f8ff56191a775e9*

3. Konstruktory, složitější volání metod (práce s grafikou v prostředí Greenfoot)

Tato kapitola je zaměřená na volání metod pomocí volání objektů prostředí Greenfoot na kreslení instancí. Studenti budou kreslit instance tříd **Wall**, **Tower** a **Player**.

V tabulce 14 je uvedené porovnaní této kapitoly v projektu Mravenci a nejvíce podobné kapitoly v projektu Bomberman, která má název algoritmus. Všimnete si, že tyto kapitoly jsou trochu odlišné, protože naše ukazuje základní práci s konstruktory a představuje grafiku v prostředí Greenfoot, zatímco v projektu Bomberman je více zaměřená na algoritmus. Proto je i z tohoto důvodu možné poukázat na rozdíly.

Tabulka 14: Porovnaní pracovní zátěže kapitol konstruktory, složitější volání metod (práce s grafikou v prostředí Greenfoot) v projektu Mravenci a podobné kapitoly v projektu Bomberman - algoritmus, ovládací prvky aplikace, vytváření metod



Bomberman	4h 30min	Ants	2h 45min

3.1. Kreslení objektů v prostředí Greenfoot – Wall

Představte studentům konstanty v Javě – definujte **wallSizeX** a **wallSizeY** jako **static final** atributy (slovo **final** zajistí, že se hodnota nedá změnit, tedy že se jedná o konstantu) s hodnotami **32** a **3**. Ke konstantě se přistupuje jako ke statické proměnné přes název třídy, tedy např. **Wall. wallSizeX**. Potom implementujte funkci **redraw()** ve třídě **Wall**, kde se vytvoří nový obrázek s danou velikostí a vyplní se jako obdélník.

Commit: *3d728f73182b79b36d95a5bb1742cd870d82f906*

3.2. Kreslení objektů v prostředí Greenfoot – Tower

To samé se opakuje i v případě třídy **Tower**. Avšak je to komplikovanější, protože věž se skládá i ze střechy, která je realizovaná jako polygon. Polygon vyžaduje pole bodů. Pokud chcete, můžete studentům podat stručné vysvětlení polí, i když pole nejsou součástí této kapitoly. Pobavte se se studenty o analogii termínů pole a seznam.

Commit: *98e55ae90a47fffd79152eabd80b6e13a15d91d5*

3.3. Definování dalších vlastností hráče

Pokuste se identifikovat další vlastnosti hráče – konkrétně jméno hráče a počet cihel, mečů a kouzel. Potom tyto vlastnosti implementujte jako atributy ve třídě **Player** a inicializujte je v konstruktoru.

Commit: *6d2b7771e8abe90117d61676215a39592ed41d39*

3.4. Vykreslení hráče

Poslední úlohou v této kapitole je vykreslení třídy **Player**. Musíte nakreslit ikony jednotlivých zdrojů, počty těchto zdrojů a též jméno hráče.

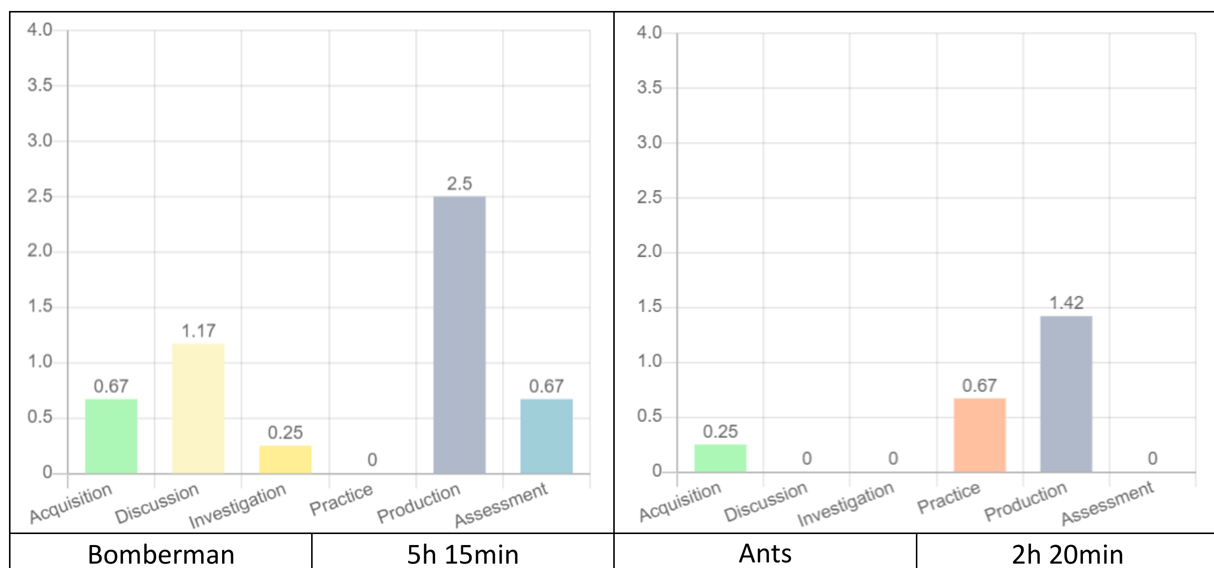
Commit: *a07482609d4494babdab16e67e221523d8cb5683*

4. Větvení, podmíněné vykonávání

Tato kapitola se zaměřuje na větvení programu a podmíněné vykonávaní častí naší hry. V této hře to je potřebné v několika případech, jako je například vykreslení prvního hráče v levé části obrazovky, druhého v pravé, atd.

Tabulka 15 ukazuje rozdíly mezi podobnými kapitolami ve hrách Bomberman a Mravenci. Jak je vidět, projekt Bomberman klade větší důraz na diskusi, než projekt Mravenci. I celková pracovní zátěž v hodinách je přibližně poloviční v porovnaní s projektem Bomberman. Je to způsobeno tím, že větvení je v tomto projektu rozdělené do více částí - tato část je spíše úvodem o základním použitím větvení.

Tabulka 15 Porovnání pracovní zátěže kapitoly větvení, podmíněné vykonávání mezi projekty Bomberman a Mravenci



4.1. Vytvoření třídy Hra

Předtím, než začnete do Vašeho kódu vkládat logiku větvení, musíte vytvořit třídu **Game**, která bude obsahovat oba hráče a v budoucnosti bude zpracovávat logiku hry, přepínaní tahů, vykonávání karet, atd. Zatím tam vložte atributy pro dva hráče. A vytvořte jejich instance v konstruktoru třídy **Game**.

Commit: *06aaf814f2adcc0cda1dd20f5993eea3fcfcbf2e*

4.2. Větvení, podmíněné vykonávání kódu – hráči jsou zobrazeni na příslušných stranách herního plánu

V tento okamžik by se měl zavést nový atribut pro třídu **Player** – informaci o tom, zda se má hráč vykreslit na levě nebo pravé straně obrazovky. Tuto informaci použijte na nastavení příslušných

vlastností hráče – pozici stěny, věže, hud (angl. head-up display) a jména. Tento posun na základě pozice hráče by se potom měl přidat do metody **redraw()**.

Commit: *c84a9065d8a5c251b2f25c61720ed8e54eb5f1d7*

4.3. Přidáváni instancí do světa

V další úloze se ve hře vytvoří počáteční zobrazení instancí tříd **Player** pro hru a ve světe se vytvoří instance hry.

Commit: *f6102678b62423d9888f7beea802129d550e19cd*

4.4. Přidaní instanci do světa 2

Aby se instance třídy **Player** správně vykreslila při vytvoření instance třídy **Game**, je potřeba přidat do světa instance tříd **Wall** a **Tower** a zavolat metodu **redraw()** v metodě **act()**. Zde si můžete vysvětlit vykonávání herní smyčky (metoda **act**).

Commit: *eb53be86180b4f3e043ea5b0363fccef559d4c58*

4.5. Podmíněné vykonání kódu jen jednou

Při pokusu o spuštění hry po poslední úloze můžete narazit na problém - objekty se do světa přidávají vícekrát za sekundu. Můžete dát studentům úlohu na odstranění tohoto problému nebo ho odstranit spolu s nimi. Jedním z možných řešení je zavedení nového atributu typu **boolean**, který by uchovával informaci o tom, zda se počáteční objekt přidaný do světa vykonal nebo ne, a po prvním vykonaní metody **act** by se táto vlastnost nastavila na **true**.

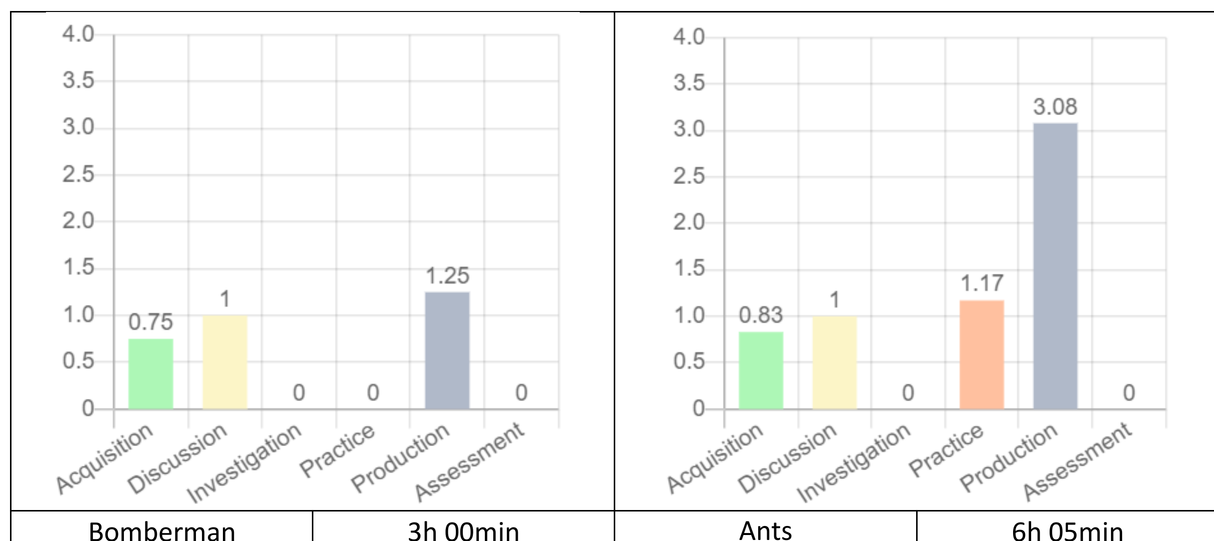
Commit: *707304650909d8450bac27a5264cdaef0e103c6a*

5. Algoritmus, výčtový typ (enum), pole

V této kapitole se rozebírají další pojmy, jako je algoritmus, výčtový typ (**enum**), pole a cyklus nad polem prvků. V této časti studenti implementují karty, které se budou používat na hraní hry.

Tabulka 16 obsahuje porovnaní podobných kapitol v projektech Mravenci a Bomberman. Jak je možné vidět, projekt Mravenci se více zaměřuje na tvorbu a procvičování a o něco více na akvizici. I celková pracovní zátěž je přibližně dvakrát větší jako u projektu Bomberman. Je to způsobeno tím, že tato kapitola obsahuje nejen práci se seznamy, ale zavádí i pojmy jako výčtový typ (**enum**) atd.

Tabulka 16: Porovnaní pracovní zátěže algoritmus, výčtový typ (enum), pole v projektu Mravenci a podobné kapitoly v projektu Bomberman - seznam a for each cyklus



5.1. Implementace třídy Card

Nejdříve můžete se studenty diskutovat o tom, jakým způsobem by finální hra fungovala a navrhnout třídu **Card**. Měli byste dojít k řešení, které bude obsahovat informace o typu karty, její požadavky, efekt a určitý popis. Potom můžete takovou třídu vytvořit jako potomka třídy **Actor**.

Commit: *928eabaebc330c865d9eb1e28d1a03453c3031ba*

5.2. Výčtové typy (enum)

V předchozí úloze jste vytvořili typ karty jako atribut ve třídě **Card**. Diskutujte se studenty o tom, jaký typ by tento atribut měl mít - **String**, **int** atd. Můžete jim představit typ výčtový typ - **enum**. Jedná se o datový typ s konečnou množinou pojmenovaných hodnot (např. pro dny v týdnu je jedná o hodnoty: pondělí, úterý, středa, čtvrtek, pátek sobota a neděle). Vytvořte s nimi výčtový typ **CardType** a zadejte jeho jednotlivé hodnoty.

Commit: *d983729a3c57d73ef2e028e39d7f067a725ba1d0*

5.3. Větvení – příkaz switch

V tomto okamžiku byste měli implementovat vykreslení karty. To je založené na typu karty, aby se karty od sebe vizuálně lišily. Můžete studentům ukázat, jak by se použil příkaz **if** a porovnat to s příkazem **switch**. Existují tři typy karet – stavební karty, útočné karty a magické karty. Na základě typu karty se vybírá její pozadí. V případě, že to použijeme příkaz **if**, kód by mohl vypadat takto:

```
if(type == BuildTower || type == BuildWall || type == IncreaseBricks) {

       background = new GreenfootImage(“building-card.png”);

} else if(type == IncreaseSwords || type == Attack)

...
```

Tento příkaz možné nahradit příkazem **switch**. Příkaz **switch** v Javě funguje tak, že se vykoná konkrétní větev, ale nezastaví vykonávání ostatních větví, dokud nezavoláte příkaz **break**. V uvedeném případě to umožňuje sloučit více typů karet do jedné skupiny a příkaz přiřazení napsat až za poslední typ karty ve skupině typů karet. Proto je následující kód

```
switch (type) {

       case BuildTower:

              background = new GreenfootImage(“building-card.png”);

              break;

       case BuildWall:

              background = new GreenfootImage(“building-card.png”);

              break;

       case IncreaseBricks:

              background = new GreenfootImage(“building-card.png”);

              break

       ...

}
```

možné zapsat i následujícím způsobem:

```
switch (type) {
        case BuildTower:
        case BuildWall:
        case IncreaseBricks:
                background = new GreenfootImage("building-card.png");
                break
        ...
}
```

Commit: *48e9df2d8eb83d6a88d0c86667c9b43497f065f2*

5.4. Pole

Objekt **Game** by měl obsahovat instance třídy **Card**. To možné vykonat zavedením třech polí typu **Card** (můžete také rozšířit ruku – tj. počet karet, které hra poskytne hráči v jednom tahu – na více z nich). Můžete vysvětlit, že by nebylo možné uložit ještě víc karet, pokud se rozhodne ruku ještě víc rozšířit a můžete také vysvětlit koncept polí. Měli byste též vytvořit instanci pole.

Commit: *2261666afe20ba205d252c8540ff6aa22177751b*

5.5. Zjednodušení vytváření instancí karet - CardFactory

Při vytváření nových instancí typu **Card** je potřeba uvést velké množství informací. Můžete se pokusit najít řešení tohoto problému. Jedním z řešení je vytvořit třídu **CardFactory**, která bude uchovávat instance pro každou kartu a implementovat metodu **clone()** ve třídě **Card**. Takto je možné vytvářet nové karty pomocí třídy **CardFactory**, konkrétně klonováním již existujících karet.

Commit: *c3135c874c2c1181b4448e338c977ed1c9fba317*

5.6. Random – Vytvoření instance náhodné karty

Po vytvoření třídy **CardFactory** a její metody **clone()** byste měli být připraveni na další část. Třída **CardFactory** by totiž měla umět vytvořit i náhodné instance třídy **Card**. Můžete vysvětlit generátor náhodných čísel a implementovat metodu, která bude klonovat (**clone()**) náhodnou kartu a také náhodnou základní kartu (základní karty jsou karty bez nákladů – to je implementované proto, abyste mohli zaručit, že hráč může vždy zahrát aspoň jednu z poskytnutých karet)

Commit: *462f2a8583b37636d4a9c6fff2ddd09520485d51*

5.7. Cyklus přes pole

V tomto okamžiku je zapotřebí ve hře vytvořit instanci třídy **CardFactory** a implementovat generování karet a jejich vymazání (odstranění ze světa) - při vysvětlování můžete zavést vybranou formu cyklu.

Commit: *fd7dae3ec277ef66986f63ae8c85481c4c9f22d3*

5.8. Vykreslení hry

Posledním úkolem v této kapitole je implementace kreslení celé hry. Během tohoto procesu byste měli připravit i karty hráčů pomocí už vytvořené metody a také zavést atribut pro uchovávání informace o tom, či je aktivní první hráč. Vykreslení hry by mělo skládat z vykreslení všech karet a vykreslení informace, jaký hráč je právě na tahu.

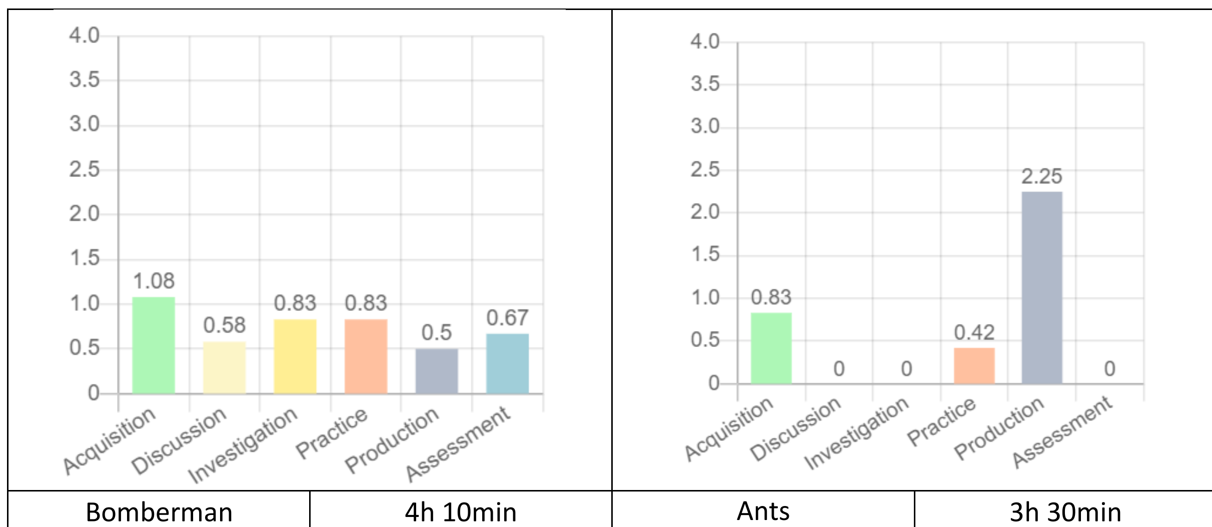
Commit: [*f7f6b68566ebe2a1c66bc284c3542e562489100b*]

6. Zpracování uživatelského vstupu, logika hry

Tato kapitola se zaměřuje na zpracování vstupu od uživatele - jak od něj získat například jméno hráče, jak zpracovat kliknutí na karty a jak dokončit logiku hry. V této kapitole jsou také představené některé pokročilé koncepty (např. singleton).

Tabulka 17 ukazuje porovnaní této kapitoly s nejvíce podobnou kapitolou v projektu Bomberman - algoritmus, ovládací prvky aplikace, vytváření metod. Upozorňujeme, že i když jde o poslední kapitolu tohoto projektu, studenti již znají mnoho pojmů zavedených v projektu Bomberman. Proto je zde výrazně menší důraz na zkoumání a diskusi jako v projektu Bomberman a větší důraz na tvorbu.

Tabulka 17: Porovnaní pracovní zátěže kapitoly zpracování uživatelského vstupu, logika hry v projektu Mravenci a podobné kapitoly v projektu Bomberman - algoritmus, ovládací prvky aplikace, vytváření metod



Bomberman	4h 10min	Ants	3h 30min

6.1. Zadávaní jmen uživatelem

V této úloze byste měli studentů vysvětlit dialogová okna (metoda **ask()** ve třídě **Greenfoot**) a nastavit názvy hráčů podle hodnot získaných ze vstupu.

Commit: [*17b9237fdabb19bb16d7c276fa06ce242dac3404*]

6.2. Statická instance třídy - Game jako singleton

Instance třídy **Game** by měli být vytvořeny jen jednou – tento problém můžete prodiskutovat se studenty a můžete poskytnout řešení ve formě singletonu – statické instance třídy **Game** a soukromého konstruktoru (**private**). Singelton (česky jedináček) se využívá v situaci kdy chceme, aby v celé aplikaci existovala pouze jedna instance určité třídy. Známým příkladem singletonu je např. schránka v operačním systému. Je to potřebné z toho důvodu, že když hráč použije kartu, měla by existovat reference na instanci třídy **Game**, aby se kliknutí na ní dalo správně zpracovat, jak se o tom bude hovořit v další časti. Jiný způsob implementace (bez singletonu) je poskytnout instanci třídy **Game** konstruktorem tříd **Card** a **CardFactory**.

Commit: [*fd4369bd7b88f15119354d7f243c9093c769a964*]

6.3. Zpracováni kliknutí na kartu

Kliknutí myší se zpracuje voláním metody **mouseClicked()** třídy **Greenfoot** v příslušné metodě **act()**. Po kliknutí byste měli zavolat metodu **useCard()** v instanci třídy **Game** a poslat odkaz na sebe (**this**).

Commit: *d1846818d19011202ed63672336b14886fede9d1*

6.4. Implementace metod pro práci s hodnotami atributů v instancích tříd Card a Player

Před implementací zbylé logiky pro hru potřebujete i další gettery a settery (setter je metoda pro nastavení hodnoty privátních atributů) pro instanci tříd **Player** a **Card**. Měli byste je implementovat (pro studenty by to neměl být problém, neboť to už bylo známé dříve).

Commit: *034563a26c04c4eef2a0dea9cc57a3c309483df4*

6.5. Implementace podpůrných metod pro hráče a oprava ve světe

Protože je nyní třída **Game** singleton, je zapotřebí ji přidat do světa ve třídě **MyWorld**. Dalším úkolem je implementovat podpůrnou metodu pro hráče, která bude sloužit na obdržení určitého množství poškození. Příslušný hráč by si měl podle toho snížit zeď nebo věž.

Commit: *e0d283ee339506ef34366a2b689aba1c17f6391c*

6.6. Implementace logiky hry

Na závěr je nutné implementovat herní logiku reprezentovanou metodou **turn()**, která se skládá z následujících kroků popsaných níže.

1. Zkontrolujte, zda jeden z hráčů vyhrál - to znamená, že pokud věž některého z hráčů dosáhne výšky 100 nebo klesne pod hodnotu 0. Pokud je splněná jedna z těchto podmínek, zobrazí se vítězná obrazovka a herní cyklus se ukončí zavoláním příkazu **return**.
2. Přepněte aktívního hráče na druhého – stačí znegovat hodnotu atributu **isPlayer1Active**.
3. Připravte karty pro hráče na tahu - již byla vytvořená metoda **prepareCards().** V tomto kroku ji stačí jen zavolat.
4. Spravujte tah hráče – konkrétně klikání na karty. Protože instance třídy **Card** dokáže reagovat na klikání, stačí zavolat metodu **draw()**, která vylosuje připravené karty.
5. Překreslete hráče – vykoná se pomocí metody **redraw()** pro každého hráče.

Potom je potřebné implementovat metodu **useCard()** ve třídě **Game** použitím příkazu **switch**. Ten by měl obsahovat větev pro každý typ karty a zpracovat její vykonání. Existuje 7 typů karet: **BuildTower**, **BuildWall**, **IncreaseBricks**, **IncreaseSwords**, **Attack**, **IncreaseMagic** a **StealBricks**. Jako příklad se ukáže řešení prvního typu karty – **BuildTower**. V tomto případě musíte zkontrolovat, zda hráč může tuto kartu zahrát (tj. počet jeho cihel je větší nebo rovný požadavkům karty), zvýšit hodnotu atributu **height** v instanci třídy **Tower** a snížit hodnotu atributu **bricksNumber** podle hodnoty na příslušné instancí třídy **Card**. Kód pro tento typ karty může vypadat např. takto:

```
if (activePlayer.getBricksNumber() >= card.getRequirements())

{

        activePlayer.increaseTowerHeight(card.getEffect());

       activePlayer.setBricksNumber(

              activePlayer.getBricksNumber() - card.getRequirements()

        );

}
```

Ostatní typy karet jsou v jistém smyslu podobné:

- **BuildWall –** musíte zkontrolovat hodnotu atributu **bricksNumber** příslušného hráče a zvýšit hodnotu atributu **height** v instanci třídy **Wall**,

- **IncreaseBricks** – nemusíte nic kontrolovat a zvýší se hodnota atributu **bricksNumber** pro daného hráče,
- **IncreaseSwords** – nemusíte nic kontrolovat a zvýší se hodnota atributu **swordsNumber** pro daného hráče,
- **Attack** – musíte zkontrolovat hodnotu atributu **swordsNumber** daného hráče a zavolat metodu **receiveDamage** neaktívního hráče,
- **IncreaseMagic** – nemusíte nič kontrolovat a zvýší se hodnota atributu **magicNumber** pro daného hráče,
- **StealBricks** – musíte zkontrolovat hodnotu atributu **magicNumber** daného hráče, snížit hodnotu atributu **bricksNumber** neaktívního hráče a navýšit hodnotu atributu **bricksNumber** aktívnímu hráči.

Je možné vytvořit i jiné typy karet - záleží na fantazii studentů. Toto jsou jen některé základné typy karet.

Nakonec musíte po zahraní karty zavolat metodu **turn()**, aby se zajistila správná logika hry.

Jako poslední bod hry byste měli implementovat vítěznou obrazovku. Její podoba je stejně jako ostatní obrázky v tomto projektu jsou Vás, takže buď si je vytvoříte spolu se studenty, nebo je necháte na nich.

Ještě je potřebné vykonat drobné úpravy kódu ve třídách **Player** a **Tower** kvůli čistotě kódu.

Commit: *9c91d4613fe0da1ff9c9960bf96fe7c27988fcf1*

4. Seznam použité literatury

[1] „Git," 1 10 2023. [Online]. Available: https://git-scm.com. [Cit. 1 10 2023].

[2] „GIT, SVN, mercurial – Google Trends," 2 10 2023. [Online]. Available: https://trends.google.com/trends/explore?cat=5&date=today%205-y&q=GIT,SVN,mercurial&hl=sk. [Cit. 2 10 2023].

[3] „GitHub: Let's build from here · GitHub," 1 10 2023. [Online]. Available: https://github.com. [Cit. 1 10 2023].

[4] „The DevSecOps Platform | GitLab," 1 10 2023. [Online]. Available: https://about.gitlab.com. [Cit. 1 10 2023].

5. Přílohy

5.1. Export návrhu vzděláváni pro projekt Bomberman

Prohlédněte si soubor LD_Bomberman.pdf

5.2. Export návrhu vzdělávaní pro projekt Tower defense

Prohlédněte si soubor LD_Tower_defense.pdf

5.3. Export návrhu vzděláváni pro projekt Mravenci

Prohlédněte si soubor LD_Ants.pdf

IMPRESUM

První vydání

Název
OOP4FUN: OBJECT ORIENTED PROGRAMMING FOR FUN – Příloha k průvodci pro současné učitele středních škol k výuce programování: Učební osnova pro výuku programování podle principů „light oop"

Editoři:
Michal Varga, Josef Rak, Dušan Savić, Zlatko Stapić

Autoři:
Peter Sedláček, Nika Kvaššayová, Jozef Kostolný, Michal Mrena, Patrik Rusnák, Peter Sobe, Ilija Antović, Miloš Milić, Tatjana Stojanović, Davor Fodrek, Lidija Kozina, Marko Mijač, Dijana Plantak Vukovac, Antonela Čižmešija, Dijana Peras, Goran Hajdin, Lea Masnec

Vydavatel:
Univerzita Pardubice
Fakulta elektrotechniky a informatiky
náměstí Čs. legií 565
530 02 Pardubice,
Česká republika

Pro vydavatele:
xx

Korektor:
xx

Překlad z anglického jazyka:
Josef Rak

Grafický design:
Dora Jovanovska, mag. inf.

Tisk:
powerprint s.r.o.
Brandejsovo nám. 1219/1,
165 00 Praha Suchdol

Náklady:
180

Místo a rok publikace:
Pardubice, září 2024.

Podpora:
Financováno Evropskou unií. Vyjádřené názory a názory jsou však pouze názory autora (autorů) a nemusí nutně odrážet názory a názory Evropské unie. Evropská unie ani orgán poskytující podporu za ně nemohou nést odpovědnost