UČEBNÁ OSNOVA PRE VÝUČBU PROGRAMOVANIA PODĽA PRINCÍPOV LIGHT OOP

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

Projekt	Object Oriented Programming for Fun
Skratka projektu	OOP4FUN
Číslo projektu	2021-1-SK01-KA220-SCH-00027903
Hlavný riešiteľ	Žilinska univerzita v Žiline (Slovakia)
Riešitelia projektu	Sveučilište u Zagrebu (Croatia) Srednja škola Ivanec (Croatia) Univerzita Pardubice (Czech Republic) Gymnazium Pardubice (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 publikovania	2023

Obsah

Zoznam obrázkov

Zoznam tabuliek

1. Informačný list

1.1. Popis predmetu

Cieľom predmetu je naučiť študentov riešiť programovacie úlohy pomocou základov objektovo orientovaného programovania (OOP) podľa paradigmy „odľahčeného“ OOP. Študenti sa naučia rozdeliť zadané úlohy medzi spolupracujúce objekty, určiť ich kompetencie a implementovať navrhnutý model. Predmet nevyžaduje predchádzajúce znalosti programovania. Vyučuje sa v programovacom jazyku Java. Predmet vysvetľuje „odľahčené“ koncepty OOP (ako je zapuzdrenie, dedičnosť alebo asociácia) na tvorbe počítačových hier, kde sa tieto koncepty jednoducho a intuitívne využívajú. Proces tvorby počítačovej hry je založený na tímovej práci a prakticky využíva vedomosti a zručnosti z iných oblastí informatiky a k nej príbuzných predmetov (práca s multimediálnym a kancelárskym softvérom). Návrh každej počítačovej hry je dostatočne otvorený na to, aby mohli študenti hru individuálne a tvorivo rozšíriť. Okrem toho návrh vedie k správnemu využitiu získaných vedomostí.

1.2. Charakteristika predmetu

Predmet je zameraný na predstavenie inovatívneho prístupu k výučbe programovania, ktorý je založený na riešení úloh pomocou paradigmy OOP. OOP je v súčasnosti dominantnou paradigmou pre vývoj aplikácií. Preto je vhodné, aby študenti mali vedomosti a zručnosti z tejto oblasti. Predmet využíva vývojové prostredie, ktoré ponúka rôzne formy editácie zdrojového kódu (bloková editácia pomocou zjednodušenej formy, ako aj reálna editácia zdrojového kódu), čo umožňuje vyučovať študentov na rôznych úrovniach predchádzajúcich technických znalostí a aktivít. Svojou jednoduchosťou a prehľadnosťou tento nástroj podporuje rýchle a intuitívne pochopenie vyučovaných tém, čo má pozitívny vplyv na študentov a ich motiváciu.

1.3. Cieľ predmetu

Prostredníctvom programovania interaktívnych hier v grafickom prostredí študent získa vedomosti a zručnosti, ktoré mu umožnia:

- identifikovať problém,
- identifikovať vhodné objekty na riešenie identifikovaného problému (dekompozícia objektov),
- navrhnúť triedy objektov, ako aj ich atribúty a metódy,
- identifikovať a správne využívať vzťahy medzi objektmi (asociácia, dedičnosť),
- navrhnúť algoritmus na riešenie problému a rozdeliť ho medzi kooperujúce objekty,
- používať prvky zdrojového kódu (vetvenie, cykly) na implementáciu navrhnutého algoritmu,
- efektívne používať prostriedky na ladenie zdrojového kódu (angl. debugging),
- vytvoriť jednoduchú aplikáciu s grafickým rozhraním v prostredí Greenfoot.

1.4. Výsledky vzdelávania

Výsledky vzdelávania dosiahnuté na tomto predmete je možné zhrnúť nasledovne:

- pochopenie základných princípov objektovo orientovaného programovania,
- pochopenie základov algoritmizácie,
- pochopenie syntaxe programovacieho jazyka Java,
- analýza vykonávania programu na základe zdrojového kódu,
- schopnosť vytvárať vlastné programy s využitím OOP.

1.5. Priestorové a technické požiadavky

Počítačová učebňa so samostatným pracovným priestorom pre každého študenta pracovným priestorom pre učiteľa. Za pracovný priestor sa považuje stôl, stolička a osobný počítač (PC). Všetky pracovné miesta by mali byť pripojené k LAN sieti s prístupom na internet (odporúčané).

PC by mal spĺňať tieto minimálne požiadavky:

- operačný systém (aspoň Microsoft Windows 7, Linux (Debian) alebo aspoň Mac OS 10.10),
- kancelársky softvérový balík s textovým, tabuľkovým a prezentačným editorom (napr. Microsoft Office, Libre Office, Open Office, ...),
- Java SE Development Kit (JDK),
- prostredie Greenfoot (aspoň verzia 3.8),
- jednoduchý grafický softvér,
- webový prehliadač (napr. Edge, Google Chrome, Mozilla Firefox, Opera, ...),
- príslušný softvér pre iný hardvér na PC.

2. Zásady učebnej osnovy

Navrhovaná učebná osnova (angl. syllabus) je navrhnutá tak, aby riešila problémy zistené vo výsledkoch projektu 1 (PR1) a výsledkoch projektu 2 (PR2) (pre viac informácií je potrebné pozrieť kapitolu "Zosúladenie výsledkov s výsledkami PR1" v správe PR2). V nasledujúcej tabuľke je uvedený pohľad na tvorbu učebných osnov, výsledky vzdelávania, učebné materiály a vyučovacie aktivity ako boli navrhnuté v PR2. Na základe týchto zistení potom bolo možné sformulovať zásady učebných osnov.

Zistenia v rámci PR2	Zásady učebných osnov formulované v PR3
Na stredných školách by sa OOP malo na začiatku predstaviť témami pokrývajúcimi základné koncepty programovania a špecifickejšie témy týkajúce sa OOP by boli vhodnejšie v samostatných kurzoch. Je veľmi dôležité zabezpečiť a podporiť výmenu informácií medzi stredoškolskými a univerzitnými učiteľmi za účasti tých zamestnancov, ktorí definujú učebné osnovy týkajúce sa programovacích zručností na všetkých úrovniach vzdelávania. Z hľadiska kurzu, alebo učiteľa a z hľadiska návrhu kurzu by sa mali využívať tieto inovatívne formy výučby či učebné metódy: zmiešané učenie (angl. Blended learning), učenie sa konaním (angl. learning by doing), riešenie problémov, spolupráca na probléme, tímová práca, učenie založené na probléme, aktívne učenie, laboratórne učenie. Okrem toho by sa rôzne formy inovatívnych prístupov mali uplatňovať na prednáškach, seminároch a laboratórnych cvičeniach. Na motiváciu študentov k programovaniu sa často používalo programovanie hier a gamifikácia vo všeobecnosti. Študentom sa páčila možnosť byť kreatívny alebo súťažiť s inými študentmi v oblasti vedomostí, ak bola podporená vhodným prostredím. Z výsledkov prehľadu literatúry sa vybrali štyri rôzne typy učenia sa prostredníctvom hier: učenie sa hraním (angl. learning by playing), učenie sa tvorbou hier, učenie sa pomocou nástrojov súvisiacich s hrami a učenie sa pomocou gamifikácie.	Učebné osnovy musia správne využívať OOP od samého začiatku podľa prístupu objekty ako prvé (angl. object first approach). Vhodná úroveň OOP pre stredné školy bola identifikovaná a formulovaná ako Light OOP (angl. Ľahké OOP). Light OOP sa dá s výhodou použiť pri tvorbe hier, pretože je jednoduché identifikovať objekty hry, ako aj kompetencie a vlastnosti objektov. Na motivovanie študentov pomocou hier je dôležité, aby sa študenti o hry zaujímali. Na splnenie tohto cieľa by sa malo vytvoriť niekoľko projektov, v ktorých sa tvoria hry s rôznymi mechanikami. Navyše niekoľko pripravených hier, ktoré sa pripravia, umožnia: • vytvoriť rôzne učebné materiály zamerané na rôzne podmienky výučby (online/fyzické, intenzívny/celoročný kurz, začiatočníci/pokročilí). Toto je kľúčová vlastnosť pre nadchádzajúce výsledky projektu, • vytvoriť jednu hru za prítomnosti učiteľa a ďalšie hry ako domáce úlohy (učiteľ bude môcť zistiť, či študenti dokážu aplikovať vedomosti v rôznych súvislostiach), • vytvoriť hru podľa zadaných pokynov s minimálnou interakciou učiteľa, aby študenti pochopili dôležitosť dobre napísaného technického opisu a použili naučené zručnosti na vytvorenie hry podľa zadania, • predstaviť nový koncept Light OOP ako sa v hre pridávajú nové mechaniky. To umožní zastaviť vývoj projektu, ak sa učiteľ rozhodne pokryť iba danú podmnožinu Light OOP kvôli špecifickým podmienkam v danom štáte.
Pri učení a vyučovaní konceptov OOP sa ukázalo, že učenie sa prostredníctvom hier má významný vplyv na zlepšenie zručností študentov pri riešení	

problémov a tiež študentom pomáha ich zapojenie do zábavného prostredia.	
Ako poukazujú viaceré výstupy PR2, hlavným cieľom by malo byť začlenenie úloh učenia a vyučovania do stimulujúcich a zábavných aktivít, ktoré budú mať pozitívny vplyv na vyššiu návštevnosť a mieru úspešného skončenia daných predmetov. To by zvýšilo záujem stredoškolákov o programovanie vo všeobecnosti a v konečnom dôsledku by viedlo k lepšiemu pochopeniu konceptov programovania a OOP. V takom prípade by študenti neboli „stratení“, keď by sa stretli s vysokoškolskými učebnými osnovami.	Projekty sa budú vytvárať na základe princípu učenia sa konaním. Snaha bude minimalizovať vysvetľovanie teórie a podporovať skúmanie, praktické a tvorivé aspekty učenia. Práca v skupinách povzbudzuje študentov, ktorí môžu mať na začiatku problémy. Ak sa projekt vyvíja v skupine, môžu sa použiť agilné metódy vývoja aj vyučovania. Pred implementáciou sa učiteľ môže rozhodnúť vykonať aj analytickú a návrhovú fázu projektu. To umožní využiť vedomosti a zručnosti študentov získané na predchádzajúcich prednáškach, či kurzoch, ako aj zapojiť študentov do tvorby projektu. Študenti môžu byť požiadaní, aby: • pracovať s informačnými zdrojmi s cieľom nájsť správnu hru, • formulovať pravidlá hry, respektíve požiadavky na ich uplatnenie (v ústnej alebo písomnej forme), • pripraviť multimediálny obsah (obrázky/zvuky).
Celkovým cieľom PR2 bolo nájsť vhodné a inovatívne nápady a prístupy k učeniu a vyučovaniu, ktoré by tieto problémy vyriešili. Ako bolo uvedené v predchádzajúcich kapitolách, existuje niekoľko vybraných osvedčených postupov, ktoré by sa mohli použiť na zlepšenie dosahovania výsledkov vzdelávania počas stredoškolského vzdelávania. Avšak je potrebné tiež poznamenať, že výsledkom prepracovania učebných osnov by malo byť zavedenie tém OOP ako aj stanovenie cieľov pri dosahovaní výsledkov vzdelávania súvisiacich s OOP. Dôležitý je aj výber vhodného typu hodnotenia: (online) dotazníky sú jedinou akceptovanou metódou na hodnotenie spokojnosti, užitočnosti, záujmu, angažovanosti a zjednodušenia konceptov programovania a OOP u študentov, ktoré budú definované na stredoškolskej a nie na univerzitnej úrovni.	So zameraním sa na princíp učenia sa konaním sme sa snažili minimalizovať aktivizačné aktivity a posilniť časť skúmania. Projekty budú postavené tak, aby umožňovali jednoduché rozšírenie, aby motivovaní študenti boli schopní pracovať na projektoch samostatne. Na overenie navrhovaných učebných osnov pripravíme pre každý projekt učebný plán. Následne porovnáme analytické výstupy nových projektov využívajúcich Light OOP s analytickými výstupmi učebného plánu vytvoreného pre projekt, ktorý bol vypracovaný v minulosti a je už nasadený v praxi na Slovensku (okrem iných škôl v krajine ho využíva aj partner projektu Obchodná akadémia Považská Bystrica) a v Českej republike (využíva ho partner projektu Gymnázium Pardubice). Pozitívne ohlasy na tento overený projekt boli zverejnené v PR2, preto predpokladáme, že pri dodržaní rovnakých osvedčených princípov a postupov prenesieme pozitívne prijatie navrhovaného učebného plánu.
Využitím tímovej práce v úlohách OOP budú študenti mať možnosť podeliť sa o svoje	Pri navrhovaní tu predstavených projektov založených na hrách sme mali na pamäti využitie

vedomosti a tak rozšíriť implementáciu základných konceptov programovania na ostatných študentov (peer-to-peer učenie).	rôznych techník, ako je napríklad EduScrum. Z tohto dôvodu má každý projekt vlastné úložisko na GIT-e a tiež sú učiteľom vysvetlené príslušné náležitosti. Hoci nie je nutne potrebné využitie tohto prístupu a učitelia môžu stále používať tradičný prístup, použitie agilných techník povedie k výučbe:

Výsledkom tohto projektu bude súbor materiálov, ktoré poskytnú vysoko motivovaným študentom s dobrými predchádzajúcimi vedomosťami možnosť rozšíriť si tieto vedomosti prostredníctvom rôznych aktivít a úloh. Takíto študenti by mohli výrazne zlepšiť celkové výsledky celej skupiny, ak dostanú príležitosť podeliť sa o vedomosti alebo viesť tímy.	• základov používania systémov riadenia verzií - odporúčame GIT ako najpoužívanejší systém riadenia verzií, pričom jeho používanie uľahčuje
	  ○ zdieľanie zdrojových kódov a
	  ○ vývoj nových funkcií pre herné projekty bez vplyvu na priebeh projektu (napr. v samostatnej vetve), čo bude motivovať ambicióznych študentov;
	• tímovej práce - každý študent bude zodpovedný za konkrétny aspekt hry, čo vedie k potrebe efektívnej komunikácie medzi členmi tímu;
	• efektívnej správy času (angl. time management) - jednotlivé časti projektov budú musieť byť dodané včas, aby ich bolo možné zlúčiť a pokračovať v ďalšej práci, avšak správne používanie systému riadenia verzií a OOP otvára mnoho možností, ako sa vysporiadať s časovými ťažkosťami, čo môže viesť k pozitívnej motivácii študentov aj v náročných situáciách.
	
	Zrealizujeme viaceré školenia s učiteľmi, ktoré budú zahŕňať úvod do GIT, ako aj využitie princípov agilného vývoja softvéru vo vyučovaní. Študenti vytvoria projektové tímy s konkrétnymi úlohami, na stand-upoch si budú vymieňať nápady, dávať a zadávať si ciele, realizovať šprinty, vytvárať dokumentáciu a ďalšie artefakty a prezentovať svoje riešenie.

Bolo by vhodné podporovať používanie viacerých rôznych nástrojov a programovacích jazykov v skorších ročníkoch štúdia. Koncepty programovania by sa mohli zjednodušiť pomocou vizualizačných nástrojov. Hoci niektoré krajiny už zaviedli používanie niektorých nástrojov, ako napríklad Logo alebo Scratch, tieto sú zaujímavé pre základné školy, ale nie pre stredné školy. Preto by sa mali používať pokročilejšie nástroje, ktoré sú určené na podporu OOP. Nástroje Alice a Greenfoot sú	Používame prostredie Greenfoot, ktoré využíva programovací jazyk Java. Java je v súčasnosti veľmi populárny a v praxi veľmi rozšírený programovací jazyk. Greenfoot tiež obsahuje blokový editor zdrojového kódu pre jazyk Stride. Toto poskytuje možnosti pre učiteľov, ktorí budú chcieť používať prezentované techniky v tomto učebnom pláne so študentmi mladšieho veku.

tými nástrojmi, ktoré sú v tejto oblasti významné.	Greenfoot je veľmi vizuálny a od začiatku umožňuje vytvoriť vizualizovaný objekt, ktorý je „živý“ a s ktorým možno interagovať. Preto je teoretický úvod minimalizovaný a študenti začnú pracovať hneď od začiatku.
Ako uviedli študenti, pri výučbe programovania je dôležité, aby učitelia používali nové a aktuálne učebné materiály a tvorivé vyučovacie metódy. Taktiež je potrebná dostupnosť a flexibilita učiteľa pri práci so študentmi mimo vyučovania, aby motivoval študentov a vyvolal u nich väčší záujem o predmet.	Na základe prezentovaných princípov sa vytvoria moderné učebné osnovy pre učiteľov, ktoré pokrývajú témy Light OOP, sú založené na projektovej práci a používajú koncept object first. Vytvorí sa niekoľko projektov založených na hrách, pričom každý z nich bude dostupný vo forme GIT repozitára. Pre každý projekt bude vytvorený učebný projekt, čo umožní overiť tieto projekty s už v praxi využívaným a pozitívne prijatým prístupom.

Obsah a rozsah vzdelávacieho programu sa líši v závislosti od projektu hry, ktorý sa počas vyučovania bude vyvíjať. Podrobnú analýzu nájdete v príslušnom vzdelávacom dizajne a v priložených súboroch analýzy.

3. Projekty

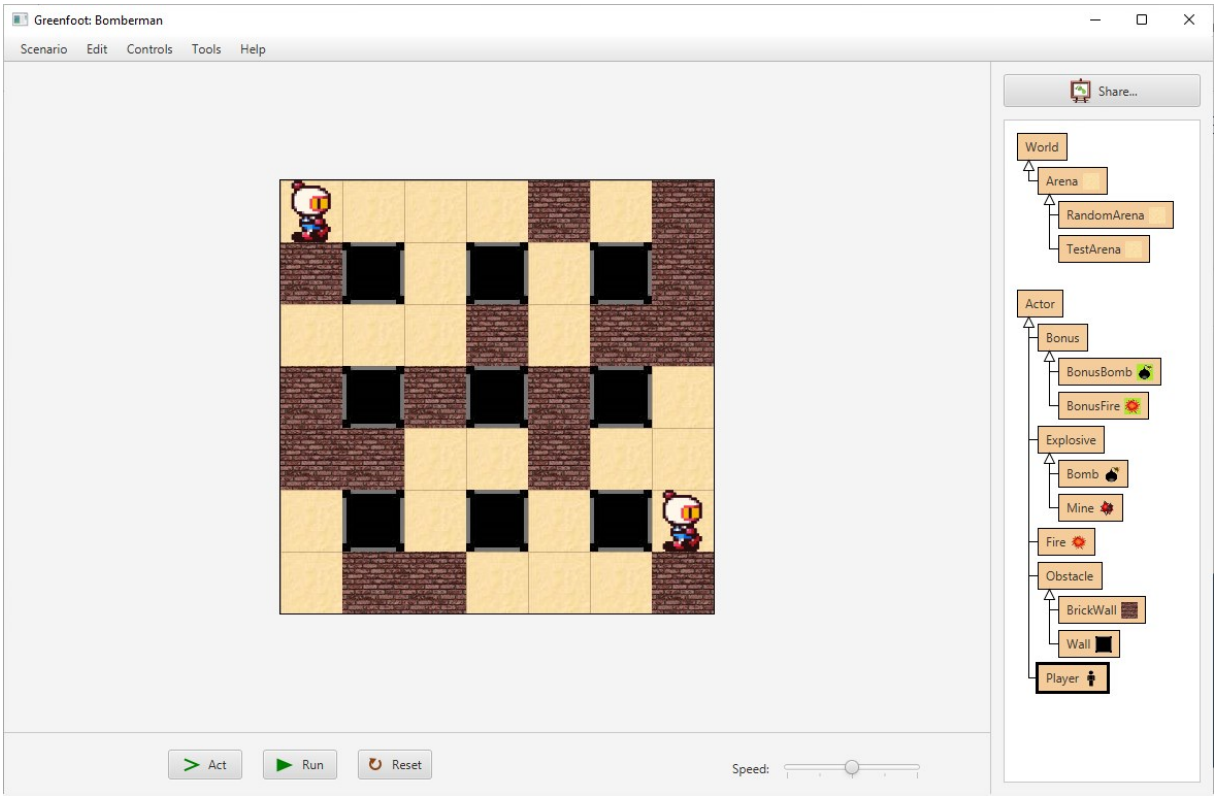
V súvislosti s princípmi učebných osnov boli vytvorené dva herné projekty, ktoré vhodne využívajú princípy object-first a učenie sa konaním. Okrem toho sme spracovali projekt Bomberman, ktorý bol nosným projektom národného projektu IT Akadémia, ktorý sa využíva v praxi na Slovensku. Tento projekt sme použili na overenie navrhovaných projektov. Organizácia všetkých kapitol projektov je nasledovná.

- **Popis projektu** – základný popis hry so snímkou obrazovky hotovej aplikácie a súhrnnými pravidlami hry. V popise projektu je uvedené aj prepojenie s témami Light OOP.
- **Odkaz na zdrojové kódy** – zdrojové kódy sú organizované formou GIT [1] repozitára. Používanie správne spravovaného repozitára privádza učiteľa k používaniu moderného prístupu v správe zdrojového kódu. Použili sme GIT ako systém riadenia verzií, ktorý patrí v posledných rokoch medzi najpopulárnejšie [2]. Použitie GIT-u tiež umožnuje použitie cloudových služieb pre správu repozitárov ako GitHub [3] či GitLab [4], ktore sú zdarma k dispozícii a ponukájú množstvo služieb pre zlepšenie tímovej spolupráce. Každý repozitár sa riadi nasledovnými zásadami:
  - **Vetva na tému** – úlohy každej témy z učebných osnov sú vypracované v špecifikovanej vetve. Hlavná vetva obsahuje len inicializačný commit (potvrdenie) a zlúčenia vetiev tém.
  - **Commit na úlohu** – každá úloha, ktorá je zameraná na tvorbu/zmenu zdrojového kódu má formu commit-u. Popis commit-u zodpovedá číslu príslušnej úlohy.
- **Prepojenie s návrhom vzdelávania** (angl. learning design) – učebné osnovy sú vytvorené vo forme návrhu vzdelávania. Návrh vzdelávania nám umožňuje definovať výsledky vzdelávania (ktoré pokrývajú potrebné kompetencie identifikované v PR1 a PR2) a priradiť ich k témam (pozri prepojenie na vetvy príslušného GIT repozitára). Témy sú usporiadané do jednotiek, ktoré sa skladajú z TLA (pozri prepojenie na commit v príslušnom GIT repozitári). Použitie návrhu vzdelávania umožňuje analyzovať rozdelenie času, čo priamo súvisí s overovaním deklarovaného princípu učenia sa konaním. Keďže návrh vzdelávania bol spracovaný aj pre už zavedený a v pedagogickej praxi používaný projekt (Bomberman), toto umožňuje identifikovať potenciálne problémy v návrhu. Aby bolo možné vykonať validáciu, projekty sú usporiadané do tém rovnakým spôsobom.
- **Pokrytie tém z light OOP.**
- **Obsah a rozsah vzdelávacieho programu** – prehľad zaťaženia študenta v konkrétnom type vzdelávania, ako aj prehľad prínosu tém k jednotlivým výsledkom vzdelávania.
- **Zoznam projektov.** Každý projekt obsahuje:
  - krátky popis,
  - porovnanie návrhu vzdelávania,
  - zoznam úloh.

3.1. Bomberman

Bomberman je pomerne známa hra pre viacerých hráčov. Hra sa odohráva v aréne, v ktorej sa nachádzajú hráči, ako aj niektoré pevné prekážky. Hráč môže klásť bomby, ktoré po určitom čase vybuchnú. Cieľom hry je eliminovať súperov pomocou bômb. Niektoré prekážky v aréne sa dajú zničiť pomocou bômb. Po zničení prekážky sa v hre môže objaviť náhodný bonus, ktorý napríklad zvyšuje rýchlosť hráča alebo silu hráčových bômb. Hra sa končí, ak v hre zostal len jeden hráč, ktorý v takom prípade vyhráva. Môže sa stať, že v hre nezostal žiadny hráč. V takom prípade sa hra končí remízou.

Projekt Bomberman pokrýva väčšinu tém ľahkého OOP. Zameriava sa na hlavné aspekty OOP, ktoré sú uvedené v nižšie uvedených témach. Navyše uvádza niektoré témy presahujúce rámec ľahkého OOP, ako je generovanie a používanie náhodných hodnôt pomocou triedy **`Random`**.



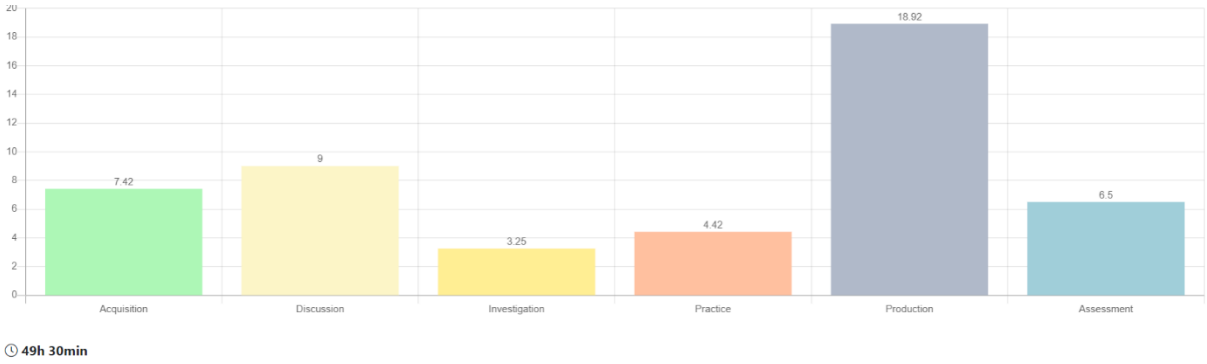
Obrázok 1: Prostredie Greenfoot s konečným stavom projektu Bomberman

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

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

3.1.1. Obsah a rozsah vzdelávacieho programu

Celková pracovná záťaž pre študenta je 49h 30min a je rozdelená nasledujúco:



Obrázok 2: Pracovná záťaž študenta pri riešení projektu Bomberman

Konštruktívne smerovanie projektu(angl. Constructive alignment) je zosumarizované v nasledujúcej tabuľke:

Tabuľka 1: Konštruktívne smerovanie 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						

Pre detailnejší plán je potrebné prejsť na kapitolu 5.1.

3.1.2. Témy

Projekt Bomberman je rozdelený do 10 tém:

Aplikované témy z light OOP sú:

- triedy, objekty, inštancie,
- metódy, odovzdávanie argumentov metód,
- konštruktory,
- atribúty,
- zapúzdrenie,
- dedičnosť,
- abstraktné triedy,
- životný cyklus projektu.

1. Úvod do prostredia Greenfoot

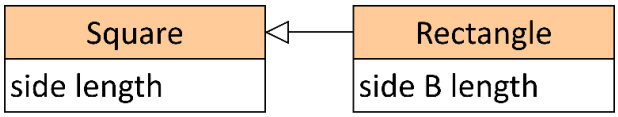
Táto téma sa venuje základnému nastaveniu projektu. Študenti sa naučia nastaviť rozmery a vzhľad prostredia, vytvoriť triedu (ako podtriedu triedy **`Actor`**), vytvoriť jej inštanciu, poslať jej správu a sledovať jej vnútorný stav.

1.1. Identifikovanie objektov

Identifikujte predmety vo svojom okolí a uveďte ich vlastnosti a činnosti, ktoré môžu vykonávať. Dokážete identifikovať objekty, ktoré nemajú žiadne vlastnosti? Dokážete identifikovať objekty, ktoré nedokážu nič robiť? Dokážete identifikovať nehmotné objekty (také, ktorých sa nemôžeme fyzicky dotknúť)?

1.2. Overenie hierarchie objektov

Na nasledujúcom obrázku je znázornená hierarchia tried **`Square`** a **`Rectangle`**. Je to dobrá hierarchia?



1.3. Vytvorenie jednoduchej hierarchie

Vytvorte hierarchiu tried pre:

a) dopravné prostriedky,
b) zvieratá
c) a časti počítača.

Uveďte vlastnosti, ktoré definuje každá trieda. Uveďte zoznam vlastností definovaných každou triedou. Ktoré triedy sú v návrhu nehmotné, takže sa ich nemôžeme dotknúť? Takéto triedy budeme v budúcnosti nazývať abstraktné.

1.4. Vytvorenie políčka

V grafickom editore vytvorte políčko, ktorá bude graficky znázorňovať bunku sveta. Vyberte štvorcové zobrazenie, ideálne 60x60 pixelov. Importujte alebo uložte obrázok do adresára projektu v adresári **`images`** (obrázky). Nastavte obrázok ako obrázok sveta. Toto sa vykoná kliknutím pravým tlačidlom myši na **`MyWorld`** a výberom možnosti **`Set image...`** . Všimnite si, že trieda **`MyWorld`** dostala v diagrame tried malú ikonu, ktorá predstavuje jej grafickú podobu.

1.5. Vytvorenie sveta

Upravte konštruktor triedy **`MyWorld`** tak, aby vytvoril svet s 25x15 bunkami, pričom každá bunka bude mať veľkosť 60 pixelov. Ako by bolo potrebné upraviť obrázok, aby svet vyzeral ako šachovnica (t. j. so striedaním rôznych farebných políčok)?

Ďalej vytvorte triedu **Player**. To vykonáte kliknutím pravým tlačidlom myši na triedu **Actor** a výberom **New subclass...**. Triedu pomenujte **Player** a vyberte jej obrázok z knižnice. Opäť si všimnite malú ikonu vedľa triedy **Player** podobnú triede **MyWorld**. Ak chcete vytvoriť inštanciu triedy **Player**, kliknite pravým tlačidlom myši na triedu **Player**, vyberte položku **new Player** a ikonu s hráčom presuňte na pozadie sveta. Kliknutím ľavým tlačidlom myši umiestnite inštanciu. Ak chcete zobraziť stav inštancie, kliknite na ňu pravým tlačidlom myši a vyberte položku **Inspect** (zobraziť).

1.6. Preskúmanie stavu hráča

Uchopte vytvorenú inštanciu triedy **Player** myšou a presuňte ju na iné miesto vo svete. Sledujte aktuálny náhľad vnútorného stavu - čo vidíte? Vytvorte ďalšiu inštanciu triedy **Player** a zobrazte aj jej vnútorný stav. Opäť jednu z týchto dvoch inštancií potiahnite myšou - ktorý vnútorný stav sa zmenil?

1.7. Interakcia s hráčmi

Zavolajte metódy poskytované prostredím Greenfoot nad rôznymi inštanciami triedy **Player**. Ak to chcete urobiť, kliknite ľavým tlačidlom myši na inštanciu a vyberte napríklad metódu **void move(int)**. Po zobrazení výzvy zadajte celé číslo. Sledujte, ako sa mení vnútorný stav inštancie.

2. Algoritmus, ovládacie prvky aplikácie, vytváranie metód

Táto téma sa zaoberá vytváraním verejných metód, ktoré pohybujú hráčom vo svete. Predstavuje tiež nástroje prostredia Greenfoot, ktoré riadia vykonávanie scenára.

2.1. Napísanie jednoduchého algoritmu

Napíšte si postup, ako sa pripravuje káva, ako sa cestuje do školy a ako sa varí obed.

2.2. Napísanie všeobecnejšieho algoritmu

Vytvorte všeobecný algoritmus prípravy horúceho nápoja. Premyslite si, aké musia byť vstupy takéhoto algoritmu, aby bol všeobecný.

2.3. Preskúmanie inštancie triedy

Preskúmajte metódy inštancie triedy **Player**. Ak to chcete urobiť, kliknite pravým tlačidlom myši na triedu a vyberte položku **Open editor**. Čo ste zistili? Analogicky k metóde **act()** pridajte metódu **makeLongStep()**.

2.4. Implementácia metódy

Do tela metódy **makeLongStep()** pridajte taký príkaz, resp. príkazy, aby sa inštancia triedy **Player** posunula o dve bunky v aktuálnom smere. Potom vytvorte viac inštancií triedy **Player** a na každej inštancii zavolajte túto metódu. Je toto správanie očakávané?

2.5. Pridanie dokumentácie

Pridajte dokumentačný komentár pre metódu **makeLongStep()**.

2.6. Pridanie ďalšej dokumentácie

Upravte dokumentačný komentár triedy **Player**. Pridajte verziu triedy a jej autora.

2.7. Prečítanie si dokumentácie

Preskúmajte okno dokumentácie.

2.8. Pridanie akcie hráča

Upravte telo metódy **act()** v triede **Player** tak, aby sa zavolala metóda **makeLongStep()**.

2.9. Preskúmanie ovládacích prvkov aplikácie

Vyskúšajte tlačidlá na ovládanie aplikácie. Vytvorte viacero inštancií triedy **Player**. Stlačte tlačidlo **Act** - čo sa stane? Stlačte tlačidlo **Run** - čo sa stane? Po prvom stlačení tlačidla **Run** stlačte tlačidlo **Pause** -

čo sa stane? Aký vplyv má **Speed** posuvník na volanie metódy **act()** po stlačení tlačidla **Run**? Čo sa stane, keď stlačíte tlačidlo **Reset**?

2.10. Pridanie ďalšej akcie hráča

Pridajte metódu do triedy **Player**, ktorá bude vykonávať pohyb inštanciou triedy **Player** do štvorca. Svoju metódu zdokumentujte. Na pohyb a otáčanie použite príslušné metódy zo základnej triedy **Actor**. Upravte metódu **act()** tak, aby sa inštancia triedy **Actor** pri jej volaní pohybovala do štvorca. Potom overte svoje riešenie spustením aplikácie.

3. Vetvenie a ovládanie hráča

Táto téma oboznamuje študentov s vetvením vo forme **if-else** príkazu a **switch** príkazu. Vetvenie sa využíva pri detekcii hrán sveta a kolízií so stenami - čo sú nové objekty pridané v rámci tejto témy.

3.1. Pohyb hráča

Upravte kód metódy **act()** v triede **Player** tak, aby sa hráč pohyboval len po stlačení klávesu **M**. Ponechajte kód zodpovedný za otočenie hráča, keď sa dostane na okraj sveta, ale myslite na jeho umiestnenie. Kedy sa môže vykonať otočenie hráča?

3.2. Sledovanie stavu hráča

Vytvorte inštanciu triedy **Player** a umiestnite ju do stredu hracej plochy. Otvorte okno s vnútorným stavom inštancie a umiestnite ho tak, aby bolo viditeľné počas behu aplikácie. Potom spustite aplikáciu a pozorujte, ako sa menia hodnoty atribútov **x** a **y** v triede **Player**. Ako sa tieto hodnoty menia pri pohybe nahor, nadol, doľava a doprava? Použite rôzne hodnoty pre metódu **setRotation()** (0, 90, 180,270) skúste nahradiť metódu **isAtEdge()** metódami **getX()** a **getY()**.

3.3. Pridanie detekcie okrajov sveta

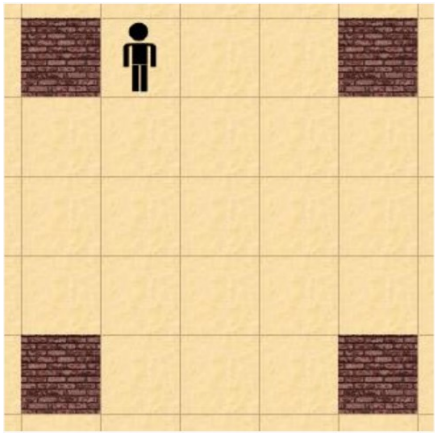
Do tela metódy **act()** pridajte kód na správne otočenie hráča po dosiahnutí dolného a ľavého okraja sveta.

3.4. Pridanie stien

Vytvorte dve nové triedy ako podtriedy triedy **Actor**. Prvou triedou bude trieda **BrickWall** a druhou triedou bude **Wall**. V grafickom editore pripravte vhodné obrázky s rozmermi 60x60 pixelov. Potom tieto obrázky priraďte k novovytvoreným triedam.

3.5. Sledovanie pohybu hráča

Vytvorte štyri inštancie triedy **BrickWall** a jednu inštanciu triedy **Player**, ako je znázornené na obrázku nižšie. Skúste odhadnúť ako sa bude hráč pohybovať. Spustite aplikáciu. Zhoduje sa váš odhad s tým, čo pozorujete?



3.6. Pridanie detekcie kolízie so stenou

Pridajte kód do metódy **act()** triedy **Player**, aby sa zabezpečilo, že sa hráč otočí o 90° proti smeru hodinových ručičiek, keď vstúpi do bunky, ktorá obsahuje inštanciu triedy **Wall**.

3.7. Predvídanie pohybu hráča

Predvídajte, ako sa bude pohybovať inštancia triedy **Player**. Súhlasí výsledok s vašou predikciou?

3.8. Pozorovanie detekcie okrajov

Umiestnite jednu inštanciu triedy **Player** do rohov sveta. Predvídajte, ako sa bude táto inštancia pohybovať po spustení aplikácie. Súhlasí výsledok s vašou predikciou?

3.9. Dokončenie riešenia kolízií so stenou

Doplňte kaskádu podmienok tak, aby sa dotyk s inštanciou triedy **BrickWall** a triedy **Wall** kontroloval len vtedy, ak sa hráč nenachádza na okraji sveta. Najprv implementujte kontrolu s inštanciou triedy **BrickWall**.

3.10. Automatický pohyb hráča

Vytvorte metódu na automatický presun inštancie triedy **Player**. Všetok kód z metódy **act()** preneste do novej metódy. Identifikátor, resp. názov metódy môže byť napríklad **moveAutomatically**.

3.11. Pohyb hráča pomocou šípok

Vytvorte metódu **moveUsingArrows()** v triede **Player**. Naprogramujte túto metódu tak, aby sa hráč pohyboval len vtedy, keď je stlačená šípka. Bude sa pohybovať v smere stlačenej šípky. Dbajte na to, aby bol kód efektívny. Túto metódu zavolajte v metóde **act()**.

3.12. Pripravenie obrázka

Pripravte pre hráča štyri obrázky, ktoré sa použijú pri pohybe hráča hore, dole, vľavo a vpravo. Rozmery obrázkov nesmú presiahnuť rozmery bunky, v našom prípade 60x60 pixelov. Pripravené obrázky umiestnite do adresára **images**.

3.13. Použitie obrázkov

Vytvorte metódu **updateImage()**, ktorá zmení obrázok hráča podľa jeho aktuálnej rotácie. Do tela metódy **act()** pridajte volanie tejto metódy.

3.14. Spustenie aplikácie

Spustite aplikáciu. Všimnite si, že obrázky sa prispôsobujú, ale otáčajú sa podľa toho, ako sa otáča hráč. Vyriešte tento problém.

3.15. Použitie viacerých hráčov

Skúste vytvoriť viacero hráčov a ovládať ich pomocou klávesnice. Čo pozorujete?

4. Premenné, výrazy a pokročilé ovládanie hráčov

Táto téma sa zaoberá zavedením premenných vo forme atribútov tried a parametrov metód. Študenti budú používať atribúty na ovládanie rýchlosti hráča, ako aj na nastavenie konkrétnych klávesov, ktoré ovládajú pohyb hráča. To umožní, aby hra mala viacero hráčov, z ktorých každý sa ovláda inými klávesmi.

4.1. Nájdenie rozdielu

Aký je rozdiel medzi nasledujúcimi algoritmami?

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. Písanie jednoduchých výrazov

Napíšte výraz pomocou logických a relačných operátorov, ktorý vyjadrí, že premenná typu **int** má hodnotu patriacu do nasledujúcich intervalov: **<-10,10)**, **(5,142)**, **(-11,-3)** alebo **(1,25>**.

4.3. Vyhodnocovanie výrazov

Vyberte hodnoty premenných **x** a **y** a doplňte ich do výrazov. Aké budú hodnoty premenných **b1** a **b2** v oboch prípadoch (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);
```

Do triedy **Player** boli pridané atribúty **String upKey**, **downKey**, **rightKey** a **leftKey**. Toto budú klávesy, ktoré sa budú používať na ovládanie pohybu inštancie triedy **Player**. Kľúčové slovo **private** znamená, že atribúty budú dostupné len v rámci triedy **Player**. V rámci tejto triedy budú atribúty prístupné prostredníctvom slova **this** (t. j. inštancia tejto triedy), t. j. **this.upKey**, **this.downKey**, **this.rightKey** a **this.leftKey**. Ďalej modifikuje metódu **moveUsingArrows()** tým, že spôsobí, že kláves **"left"** bude nahradený atribútom **this.leftKey**. Ostatné klávesy sú nahradené podobne. Klávesy na ovládanie pohybu danej inštancie musia byť špecifikované pri jej vytváraní. Na tento účel sa musí vytvoriť konštruktor. Ide o špeciálnu metódu na vytvorenie inštancie danej triedy, ktorá sa vykoná pri jej vytvorení. Konštruktor má opäť štyri parametre - **upKey**, **downKey**, **rightKey** a **leftKey**. Vo vnútri konštruktora je potrebné rozlišovať medzi **leftKey** a **this.leftKey**. Prvý je parametrom konštruktora a druhý je atribútom inštancie triedy. Konštruktor nie je v rámci triedy povinný. Ak ho neimplementujeme, použije sa predvolený s prázdnym kódom.

4.4. Testovanie konštruktora

Otestujte konštruktor a upravenú metódu na ovládanie pohybu hráča vložením dvoch inštancií triedy **Player** do sveta. Pre každú inštanciu v dialógovom okne nastavte rôzne klávesy na ovládanie jej pohybu. Otestujte, či sa vložení hráči dajú ovládať nezávisle.

4.5. Premenovanie triedy

Premenujte triedu **MyWorld** na **Arena**. Všimnite si, že názov konštruktora musí byť rovnaký ako názov triedy. Otestujte, ako sa prostredie Greenfoot správa, ak tomu tak nie je.

4.6. Pridanie ďalšieho hráča

Pridajte do sveta ďalšieho hráča, napríklad pomocou referenčného atribútu **player2**, ktorý bude ovládaný klávesmi **w** - hore, **s** - dole, **d** - vpravo a **a** - vľavo. Umiestnite hráča na súradnice **[24,14]**. Po pridaní otestujte ovládanie ďalšieho hráča.

4.7. Referenčné atribúty

Čo by sa stalo, keby sme po vytvorení oboch hráčov vykonali priradenie **`this.player1 = this.player2;`**? Bol by to problém?

4.8. Rozšírenie triedy hráča

Rozšírte triedy **`Player`** o ďalší atribút typu **`int`**, ktorý predstavuje veľkosť kroku hráča. Tu sa používajú dva konštruktory, ktoré sa líšia počtom parametrov. Ide o takzvaný preťažený konštruktor. Vždy sa použije ten so zodpovedajúcimi parametrami.

4.9. Integrovanie veľkosti kroku

Upravte metódu **`moveUsingKeys()`** tak, aby rešpektovala nový atribút dĺžky kroku. Vytvorte novú inštanciu triedy **`Player`** a otestujte funkčnosť programu.

4.10. Umožnenie hráčom pohybovať sa rôznou rýchlosťou

Vašou úlohou je zabezpečiť, aby sa hráč pohyboval po jednom políčku, ale rôznou rýchlosťou. Každý hráč môže mať inú rýchlosť. Nápovedou pre riešenie je možnosť naprogramovať rôzne rýchlosti pohybu napríklad tak, že sa hráč nebude pohybovať pri každom zistení stlačenia klávesu (detekcia sa vykonáva v metóde **`act()`**, takže ak podržíte kláves, zistíte jeho stlačenie pri každom vykonaní), ale len pri každom N-tom stlačení. Čím väčšie je N, tým nižšia bude rýchlosť hráča. Ako zadáte N? Kde ho uložíte? Ako zistíte, koľko stlačení klávesov sa už vykonalo? (Nápoveda: je potrebné aj počítadlo).

5. Spolupráca objektov a tried

Hlavným cieľom tejto témy je spolupráca objektov. V tejto téme študenti do projektu pridajú interakciu medzi objektmi v aréne - napríklad zabezpečia, aby hráč nemohol prechádzať cez steny. Okrem toho v tejto téme študenti do projektu pridajú aj objekt bomby - jednu z hlavných častí hry Bomberman.

5.1. Pridanie kódu pre vertikálny pohyb

Analogicky pridajte kód pre smer hore a dole rovnakým spôsobom (takže zmeníte hodnotu lokálnej premennej **`y`**).

5.2. Overenie možnosti pohybu

Upravte metódu, ktorá zabezpečuje pohyb hráča (**`moveUsingArrows`**) tak, aby sa pred zmenou polohy hráča overila možnosť vstupu do cieľovej bunky.

5.3. Zohľadnenie múrov

Upravte metódu **`canEnter()`** tak, aby hráč reagoval aj na inštancie triedy **`BrickWall`** a nemohol cez ne prejsť.

5.4. Pridanie bomby

Vytvorte novú triedu **`Bomb`** a navrhnite jej atribúty, ktoré budú reprezentovať silu výbuchu. Vytvorte parametrický konštruktor a inicializujte atribúty objektu.

5.5. Kontrola možnosti umiestnenia bomby

Pridajte do triedy **`Player`** metódu **`canPlantBomb()`** s návratovou hodnotou typu **`boolean`**, ktorá vráti logickú hodnotu hovoriacu o tom, či je možné umiestniť bombu do bunky, v ktorej hráč práve stojí. Bombu je možné umiestniť, ak je stlačený príslušný kláves a na bunke nie je žiadna iná bomba.

5.6. Pridanie zvukových efektov

Rozšírte hru tak, aby výbuch bomby sprevádzal zvukový efekt. Zvuk je možné nahrať alebo stiahnuť z internetu. Príkaz na prehrávanie zvuku nájdete v dokumentácii triedy **`Greenfoot`**.

6. Dedičnosť a cyklus for

Táto téma sa zaoberá základnými princípmi dedičnosti. Študenti vytvoria predka pre triedy **`Wall`** a **`BrickWall`**. Potom vytvoria testovaciu arénu ako potomka triedy **`Arena`**. Nakoniec sa táto časť zameriava na cykly s pevným počtom opakovaní.

6.1. Pridanie triedy predka

Vytvorte triedu **`Obstacle`**. Ktorá trieda je predkom triedy **`Obstacle`**? Upravte hlavičky tried **`BrickWall`** a **`Wall`** tak, aby boli potomkami triedy **`Obstacle`**.

6.2. Zjednodušenie testu obsadenosti

Po pridaní predka **`Obstacle`** je jednoduchšie testovať, či hráč môže vstúpiť do danej bunky. Vo svojej metóde **`getObjectsAt()`** vyžaduje svet ako tretí parameter triedu, ktorú má hľadať na danej bunke. Keďže **`BrickWall`** aj **`Wall`** sú **`Obstacle`**, je možné s nimi zaobchádzať jednotne. Upravte metódu **`canEnter()`** v triede **`Player`** tak, aby používala len jeden zoznam prekážok.

6.3. Úprava triedy Arena

Upravte triedu **`Arena`** tak, aby jej konštruktor obsahoval dva parametre reprezentujúce šírku a výšku. Upravte volanie konštruktora predka v triede **`Arena`** tak, aby preberal tieto parametre. Všimnite si, že **`Arena`** nemôže byť automaticky skonštruovaná v prostredí **`Greenfoot`**, pretože potrebuje parametre pre konštruktor. Odstráňte z tohto konštruktora kód zodpovedný za vytváranie a umiestňovanie hráčov do arény, to budú robiť potomkovia. Z triedy **`Arena`** môžete tiež odstrániť deklaráciu atribútov typu **`Player`**.

6.4. Oprava triedy TestArena

Upravte konštruktor triedy **`TestArena`** tak, aby vytvoril prázdnu arénu s veľkosťou 7x7 políčok. Na overenie vytvorte inštanciu triedy **`TestArena`** – z kontextového menu triedy **`TestArena`** vyberte položku **`new TestArena()`**.

6.5. Pridanie informácie na rozmer

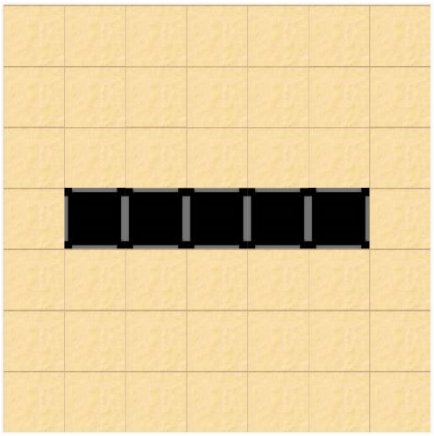
Pridajte metódu **`showDimensions()`** do triedy **`TestArena`**, ktorá vypíše rozmery arény na obrazovku.

6.6. Pridanie ďalšej informácie na rozmer

Do triedy **`Player`** pridajte metódu **`showDimensions()`**, ktorá zobrazí rozmery arény, ak sa nachádza v testovacej aréne – triede **`TestArena`**. Použite metódu **`showDimensions()`** triedy **`TestArena`**.

6.7. Pridanie stien do testovacej arény

Upravte konštruktor triedy **`TestArena`** tak, aby vytvoril arénu s rozmermi 7x7 buniek, ktorej steny sú rozložené takto:



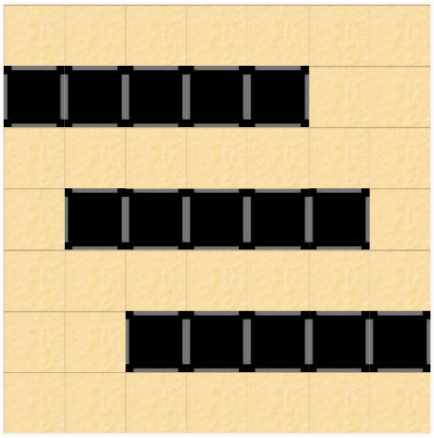
Pripomeňme si, že na vloženie inštancie triedy **Actor** do sveta (t. j. do potomkov triedy **World** a v našom prípade triedy **Arena**) môžeme použiť metódu **addObject()** triedy **World**, ktorá má tri parametre:

- objekt, ktorý sa má vložiť a je potomkom triedy **Actor**,
- x-ová súradnica bunky, ktorá sa má vložiť (t. j. index stĺpca číslovaný od 0),
- y-ová súradnica bunky, ktorá sa má vložiť (t. j. index riadku číslovaný od 0).

Pozícia **[0;0]** vo svete je vľavo hore. Keďže chceme pridať päť inštancií triedy **Wall** naraz, použijeme cyklus **for**, ktorý opakuje príkazy vo vnútri bloku pre všetky **i=1,2,3,4,5** v našom prípade. Pripomeňme si, že **super** je volanie konštruktora rodičovskej triedy, v našom prípade triedy **World**. **Super** musí byť v konštruktore na prvom mieste

6.8. Pridanie ďalších stien

Upravte konštruktor triedy **TestArena** tak, aby boli bunky rozložené podľa obrázka.



6.9. Zamyslenie sa nad tým, ako znázorniť viacero stien

Zamyslime sa nad tým, koľko informácií potrebujeme na to, aby sme mohli vytvoriť ľubovoľný riadok po sebe idúcich stien.

6.10. Pridanie metódy na vytvorenie riadku stien

Vytvorte metódu **createRowOfWalls()** v triede **Arena**, ktorá bude mať tri parametre:

- riadok (horný riadok má index 0), na ktorom sa majú začať vytvárať steny,
- stĺpec (ľavý stĺpec má index 0), od ktorého sa majú začať vytvárať steny,
- číslo vyjadrujúce, koľko po sebe idúcich stien sa má vytvoriť.

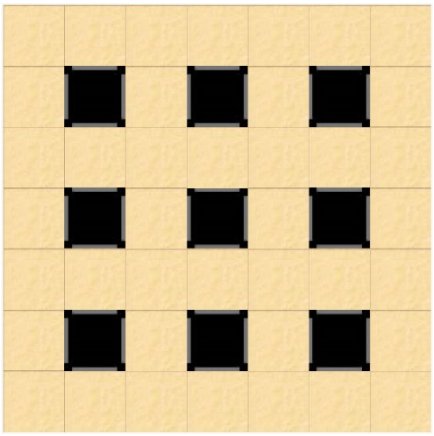
Metóda nemá žiadnu návratovú hodnotu (v takomto prípade používame kľúčové slovo **void**).

6.11. Používanie novej metódy

Upravte kód v konštruktore triedy **TestArena** tak, aby používal metódu **createRowOfWalls()** z predka – triedy **Arena**.

6.12. Zmena rozmiestnenia stien

Upravte konštruktor triedy **TestArena** tak, aby vytvoril arénu znázornenú na obrázku nižšie. Na tento účel upravte metódu **createRowOfWalls()** tak, aby obsahovala štvrtý parameter definujúci rozostupy medzi stenami.



6.13. Zamyslenie sa nad tým, ako znázorniť obdĺžnik stien

Zamyslime sa nad tým, koľko a akých informácií potrebujeme na to, aby sme mohli vytvoriť steny v obdĺžnikovom usporiadaní, ktorého počiatočný bod možno určiť a pre ktoré možno nastaviť vzdialenosti medzi stenami v riadkoch aj stĺpcoch.

6.14. Pridanie metódy vytvárajúcej obdĺžnik stien

V triede **`Arena`** vytvorte metódu **`createRectangleOfWalls()`**, ktorá bude mať nasledujúce parametre:

- riadok (vrchný riadok má index 0), od ktorého sa majú začať vytvárať steny,
- stĺpec (ľavý stĺpec má index 0), od ktorého sa majú začať vytvárať steny,
- počet riadkov, ktoré sa majú vytvoriť,
- počet po sebe idúcich stien, ktoré sa majú vytvoriť v riadku,
- počet prázdnych buniek (riadkov) medzi riadkami,
- počet prázdnych buniek medzi stenami v riadku.

Metóda nemá žiadnu návratovú hodnotu.

6.15. Používanie novej metódy

Upravte konštruktor triedy **`TestArena`** tak, aby používal metódu **`createRectangleOfWalls()`** na vytvorenie arény podľa úlohy 6.14.

6.16. Testovanie arény

Vytvorte v aréne niekoľko stien a pridajte dvoch hráčov. Otestujte funkčnosť hry, t. j. či hráči nevstúpia do bunky s objektom triedy **`BrickWall`** alebo **`Wall`**.

7. Zoznam a for each cyklus

Táto kapitola sa podrobnejšie zameriava na zoznamy, ktoré sa použijú na sledovanie daných objektov v aréne. Tiež predstavuje základné metódy práce so zoznamom (vytvorenie, pridanie prvku, odstránenie prvku, prístup k prvku) a tiež učí, ako používať cyklus for each na jednoduchý prístup ku všetkým prvkom zoznamu.

7.1. Overenie ukončenia hry

Vytvorte v triede **`Arena`** bezparametrickú metódu **`isGameEnded()`**, ktorá zistí, či hra skončila (zostal len jeden alebo žiadny hráč), a vo forme návratovej hodnoty typu **`boolean`** oznámi, či sa tak stalo. Zatiaľ predpokladajme, že koniec hry nikdy nenastane.

7.2. Ukončenie hry

Pridajte metódu **`act()`** do triedy **`Arena`**. V tejto metóde skontrolujte, či hra skončila (pomocou metódy **`isGameEnded()`**) a ak áno, ukončite hru. Ak chcete zastaviť prostredie Greenfoot, použite príkaz **`Greenfoot.stop();`**.

7.3. Pridanie zoznamu hráčov

Pridajte atribút **`listOfPlayers`** typu **`LinkedList<Player>`** do triedy **`Arena`**. Nezabudnite, že musíte importovať balík s triedou **`LinkedList`**. Inicializujte atribút v konštruktore triedy **`Arena`**.

7.4. Registrácia hráčov

Do triedy **`Arena`** pridajte metódu **`registerPlayer()`**, ktorá bude mať jeden parameter typu **`Player`** a vloží ho na koniec zoznamu **`listOfPlayers`** pomocou metódy **`add()`**. Upravte potomkov triedy **`Arena`** tak, aby zaregistrovali hráča v predkovi (**`Arena`**), keď je hráč vložený do sveta na správne miesto.

7.5. Odregistrovanie a odstránenie hráča

Do triedy **`Arena`** pridajte funkciu **`unregisterAndRemovePlayer()`**, ktorá bude mať jeden parameter typu **`Player`**. Metóda odstráni hráča zo zoznamu hráčov a potom ho odstráni zo sveta.

7.6. Správne ukončenie hry

Implementujte telo metódy **`isGameEnded()`** tak, aby metóda vrátila **`true`**, keď v hre zostane jeden hráč alebo už neostane žiaden hráč. Použite vhodné metódy zoznamu.

7.7. Spravenie bômb nebezpečnými

Upravte kód v metóde **`act()`** triedy **`Bomb`** tak, aby pred odstránením bomby vo svete zabila všetkých hráčov, ktorí sú od nej vzdialení najviac o daný modifikátor sily. Metóda **`getObjectsInRange()`** vracia zoznam typu **`List`**. Jej prvým parametrom je rozsah - v tomto prípade modifikátor sily. Jej druhý parameter je totožný s triedou, ktorej inštanciu hľadá v danom rozsahu.

7.8. Odstránenie zasiahnutých hráčov

Pomocou cyklu **`for`** prechádzajte cez všetkých hráčoch v zozname hráčov, ktorých sa bomba dotkla. Zrušte registráciu takýchto hráčov v triede **`Arena`**.

7.9. Úprava triedy Player

Vytvorte metódu **`hit()`** v triede **`Player`**, ktorá bude vyvolaná bombou hráča po tom, ako ho bomba zasiahne. V tejto metóde odhláste hráča zo sveta. Upravte kód v metóde **`act()`** triedy **`Bomb`** tak, aby odrážal novú funkcionalitu.

7.10. Odstránenie vlastníka

Vytvorte metódu **`removeOwner()`** v triede **`Bomb`**, ktorá nastaví jej atribút **`owner`** na **`null`**. Inštanciu objektu si možno predstaviť ako ukazovateľ - "šípku" na objekt. Nastavením na **`null`** ukazovateľ neukazuje na žiadny objekt.

7.11. Pridanie zoznamu bômb

Vytvorte atribút **`listOfActiveBombs`** typu **`LinkedList<Bomb>`** v triede **`Player`**. Inicializujte ho v správnom konštruktore. Upravte telá nasledujúcich metód podľa nasledujúcich pravidiel:

- v metóde **`act()`** zaregistrujte novovytvorenú bombu do zoznamu **`listOfActiveBombs`**;
- v metóde **`bombExploded()`** odstráňte bombu, ktorá bola zadaná ako parameter (tá, ktorá vybuchla), z atribútu **`listOfActiveBombs`**;
- v metóde **`hit()`** použite cyklus for each na odstránenie vlastníka zo všetkých bômb v atribúte **`listOfActiveBombs`**.

8. Súkromné metódy a cyklus while

Táto téma predstavuje cyklus while - cyklus, ktorý sa opakuje, kým sa podmienka definovaná na začiatku cyklu vyhodnotí ako **`true`**. Okrem toho učí študentov vytvárať súkromné metódy - metódy, ktoré môže zavolať len inštancia triedy, v ktorej je metóda definovaná.

8.1. Vytvorenie triedy Fire

Vytvorte triedu **`Fire`**. Vyberte vhodné grafické znázornenie. Konštruktor tejto triedy má jeden parameter, ktorý určuje, ako dlho bude oheň horieť na mieste. Zabezpečte, aby po danom čase oheň zo sveta zmizol.

8.2. Založenie ohňa

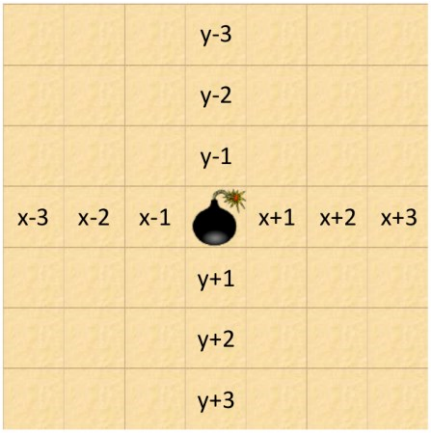
Upravte existujúci kód výbuchu bomby tak, aby na mieste výbuchu bomby zostala inštancia triedy **`Fire`**. Otestujte svoje riešenie.

8.3. Rozšírenie ohňa

Pomocou cyklu for rozšírte výbuch bomby (vytvorte inštanciu triedy **`Fire`**) smerom vpravo od bomby. Rozšírte výbuch na toľko buniek, koľko udáva atribút sily bomby.

8.4. Rozšírenie ohňa do všetkých smerov

Nastavte výbuch bomby tak, aby vytvárala ohne vo všetkých smeroch (hore, dole, vpravo a vľavo). Pomôžte si zmenou súradníc, ako je znázornené na obrázku nižšie.



8.5. Prepísanie cyklov

Prepíšte všetky cykly for pre šírenie ohňa pomocou cyklu while. Druhú časť podmienky (je možné umiestniť oheň do ďalšej bunky) zatiaľ vynechajte.

8.6. Použitie súkromnej metódy

Na rozšírenie ohňa po výbuchu bomby použite súkromnú metódu **`spreadFire()`**.

8.7. Pridanie ďalšej súkromnej metódy

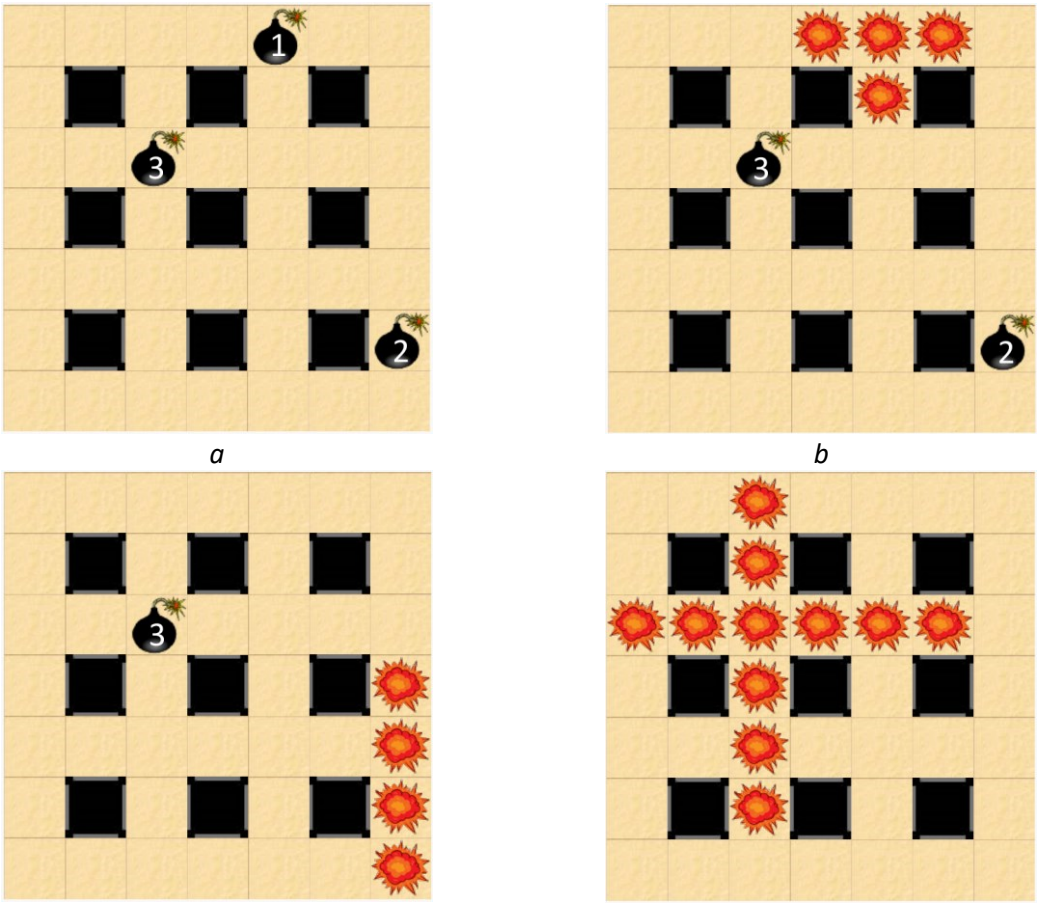
Vytvorte súkromnú metódu **`canCellExplode()`** v triede **`Bomb`**, ktorá ako parametre potrebuje súradnice riadkov a stĺpcov a vráti hodnotu **`true`**, ak v danej bunke môže nastať explózia, inak **`false`**. Šírenie ohňa nemôže pokračovať, ak:

- dosiahol okraj sveta,
- ak bunka obsahuje stenu.

8.8. Použitie súkromných metód

Pomocou metódy **`canCellExplode()`** môžete teraz upraviť podmienku v cykle while v metóde **`spreadFire()`** v triede **`Bomb`**. Upravte podmienku v cykle tak, aby rešpektovala výsledok kontroly z

metódy **`canBombExplode()`**. Otestujte funkčnosť riešenia s bombami rôznej sily medzi stenami. Testy rôznych výbuchov sú znázornené na nasledujúcich obrázkoch:



a *b* *c* *d*

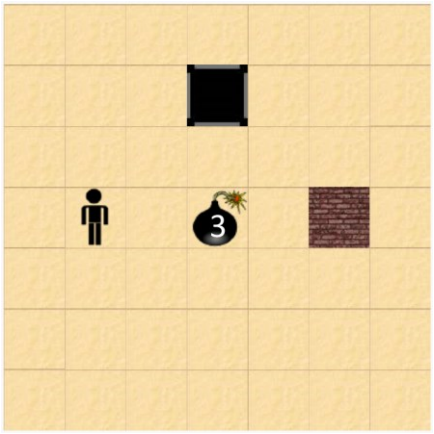
V časti a vidíme pôvodné rozloženie, v časti b vybuchla bomba so silou 1, v časti c vybuchla bomba so silou 2 a v časti d vybuchla bomba so silou 3.

8.9. Kontrola prekážky výbuchu

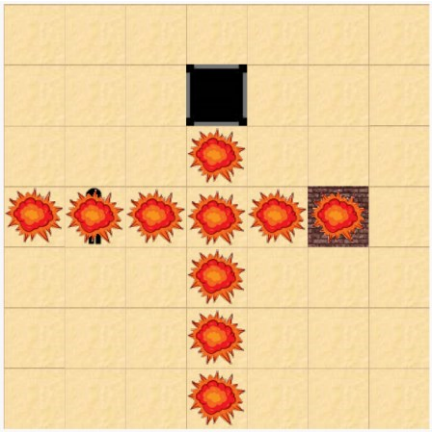
Do triedy **`Bomb`** pridajte súkromnú metódu **`canExplosionContinue()`**, ktorá dostane ako parametre súradnice riadkov a stĺpcov a vráti hodnotu **`true`**, ak bunka nezastavila výbuch na daných súradniciach. Ak bunka zastavila výbuch, metóda vráti **`false`**. Výbuch nemôže pokračovať, ak narazil na stenu.

8.10. Obmedzenie výbuchu

Pomocou metódy **`canExplosionContinue()`** upravte metódu **`spreadFire()`** v triede **`Bomb`**. Ak môže výbuch pokračovať z danej bunky, zvýšte hodnotu premennej **`i`** o 1 a prepočítajte súradnice nového riadku a stĺpca výbuchu. V opačnom prípade umelo zvýšte hodnotu premennej **`i`** na hodnotu väčšiu, ako je sila bomby, čím sa cyklus zastaví. Otestujte svoje riešenie na situácii z nasledujúceho obrázka:



A



b

8.11. Úprava kontroly prítomnosti ohňa

Upravte správanie inštancie triedy **`Bomb`** tak, aby nevolala metódu **`hit()`** hráčov vo svojom dosahu pri výbuchu. Namiesto toho bude hráč sám kontrolovať, či sa streľba neprekrýva. Upravte správanie metódy **`act()`** hráča tak, aby najprv zistilo, či sa hráč neprekrýva s inštanciou triedy **`Fire`**. Ak áno, sám zavolá svoju metódu **`hit()`**. Tým sa zabezpečí, že hráč bude zasiahnutý aj ohňom, ktorý horí po výbuchu bomby.

8.12. Pridanie série výbuchov

Upravte metódu **`act()`** triedy **`Bomb`** tak, aby bomba vybuchla aj v situácii, keď sa bomba nachádza v tej istej bunke ako oheň. Riešenie overte vykonaním reťazovej reakcie niekoľkých bômb.

9. Polymorfizmus

Cieľom tejto témy je naučiť študentov vytvárať virtuálne metódy a podľa potreby ich prekrývať v potomkoch. Taktiež sa študenti oboznámia s novou viditeľnosťou atribútov a metód – modifikátorom prístupu **`protected`**. Študenti využijú polymorfizmus na zjednodušenie existujúceho kódu.

9.1. Pridanie mín

Začnite pridaním míny. Mína vybuchne, keď na ňu hráč stúpi. Mína tiež vždy vybuchne, keď ju zasiahne oheň (napr. z bomby, ktorá vybuchla v blízkosti). Na svojom mieste zanecháva oheň (a len tam). Pridajte možnosť, aby hráč mohol položiť míny (podobne ako bomby) po stlačení klávesu (napr. **`control`** alebo **`shift`**). Podobne ako pri bombách, aj tu má hráč obmedzený počet mín (t. j. ak položí všetky míny, ďalšiu mínu môže položiť len vtedy, ak jedna z predtým položených mín vybuchne). Počiatočný počet mín je nastavený parametrom konštruktora triedy **`Player`**. Ak chcete v triede **`Player`** zaregistrovať míny a reagovať na ich výbuch, postupujte rovnako ako pri bombách (vytvorte zoznam mín, pridajte metódy **`mineExploded()`**, **`canPlantMine()`** atď.)

9.2. Pridanie predka

Vytvorte spoločného predka pre triedy **`Bomb`** a **`Mine`** – triedu **`Explosive`**. Ktoré atribúty a metódy by sa mali presunúť do predka a ktoré by mali zostať v potomkoch? Upravte existujúce triedy podľa svojho návrhu.

9.3. Úprava viditeľnosti atribútu

Upravte viditeľnosť atribútu **`owner`** v triede **`Explosive`** na **`protected`**. Viditeľnosť atribútu **`private`** by umožnila jeho použitie len vo vnútri triedy **`Explosive`** a nie v jej potomkoch. Viditeľnosť **`protected`** znamená viditeľnosť v rámci balíka, čo je v našom prípade v rámci projektu Bomberman.

9.4. Pridanie textového výstupu

Vytvorte metódu **`printWhoYouAre()`** bez parametrov v triede **`Explosive`**, ktorá nemá návratovú hodnotu. Metóda vypíše text "EXPLOSIVE" na obrazovku, kde sa výbušnina práve nachádza. Vytvorte inštanciu triedy **`Mine`** a zavolajte metódu **`printWhoYouAre()`**. Čo sa stane? Vytvorte inštanciu triedy **`Bomb`** a zavolajte metódu **`printWhoYouAre()`**. Čo sa stane v tomto prípade?

9.5. Zlepšenie textového výstupu

Vytvorte metódu **`printWhoYouAre()`** v triede **`Mine`** s rovnakou hlavičkou ako v triede **`Explosive`** (t. j. metóda bude mať rovnaký názov, rovnaké parametre a rovnaký typ návratovej hodnoty). Metóda vypíše na obrazovku text "MINE". Opäť vytvorte inštanciu triedy **`Mine`** a triedy **`Bomb`**. Skúste uhádnuť, čo sa stane, keď zavoláte testovaciu metódu v inštancii triedy **`Mine`** a inštancii triedy **`Bomb`**. Potom zavolajte metódy. Zodpovedá váš odhad výsledku?

9.6. Dokončenie textového výstupu

Prekryte metódu **`printWhoYouAre()`** v triede **`Bomb`** tak, aby na obrazovku vypisovala text "BOMB". Overte správnosť svojho riešenia.

9.7. Pridanie spracovania výbuchu

Vytvorte metódy **`shouldExplode()`** a **`explosion()`** v triede **`Explosive`** a metódu **`explosiveExploded()`** v triede **`Player`**, ako je opísané vyššie. Telá metód zatiaľ neimplementujte. Ak je potrebná návratová hodnota, vráťte **`false`**.

9.8. Umožnenie explózie výbušniny

Napíšte telo metódy **`act()`** v triede **`Explosive`**.

9.9. Umožnenie výbuchu bomby

Prekryte metódy **`shouldExplode()`** a **`explosion()`** v triede **`Bomb`**. Použite príslušný kód z metódy **`act()`** v triede **`Bomb`**. Všimnite si, že je možné jednoducho napísať telá metód, pretože nie je potrebné zdôvodňovať podmienky (to urobil predok).

9.10. Umožnenie výbuchu míny

Prekryte metódy **`shouldExplode()`** a **`explosion()`** v triede **`Mine`**. Použite príslušný kód z metódy **`act()`** v triede **`Mine`**. Prečo je potrebné na konci odstrániť metódu **`act()`**?

9.11. Pridanie interakcie s ohňom

Upravte telo metódy **`shouldExplode()`** v triede **`Explosive`** tak, aby metóda vrátila **`true`**, ak sa inštancia dotkne inštancie triedy **`Fire`**. Upravte nadradené metódy **`shouldExplode()`** v triedach **`Bomb`** a **`Mine`** tak, aby používali funkčnosť metódy predka.

9.12. Zjednodušenie atribútov hráčov

Odstráňte atribúty **`listOfActiveBombs`** a **`listOfActiveMines`** z triedy **`Player`**. Pridajte do triedy **`Player`** jeden atribút typu **`listOfActiveExplosives`** typu **`LinkedList<Explosive>`**. Inicializujte ho v konštruktore a odstráňte inicializáciu pôvodných atribútov z konštruktora.

9.13. Zjednodušenie zaobchádzania s výbuchom

Implementujte telo metódy **`explosiveExploded()`**. Použitie operátora **`instanceof`** na určenie, či je výbušnina **`Bomb`** alebo či je výbušnina **`Mine`**. Na základe jej skutočného typu zvýšte počítadlo dostupných bômb alebo počítadlo dostupných mín. Nezabudnite odstrániť výbušninu zo zoznamu aktívnych výbušnín. Nakoniec odstráňte nepotrebné metódy **`bombExploded()`** a **`mineExploded()`**.

10. Náhodné čísla

Táto téma sa venuje náhodnosti. Študenti sa oboznámia s triedou **`Random`**. Pomocou jej inštancií budú generovať náhodné čísla. Téma tiež ukazuje spôsob generovania náhodných čísel bez použitia triedy **`Random`**, a to priamo pomocou prostredia **`Greenfoot`**. Študenti použijú náhodné čísla na náhodné usporiadanie arény a na pridávanie bonusov do sveta – špeciálnych prvkov, ktoré sa vytvoria po výbuchu múru a ktoré zlepšujú vybrané vlastnosti hráčov.

10.1. Zamyslenie sa nad náhodnosťou

Premýšľajte o tom, čo je to náhodnosť, ako môžeme získať nejaký náhodný výsledok z experimentu a aké náhodné javy pozorujeme vo svete okolo nás.

10.2. Zamyslenie sa nad generovaním náhodných hodnôt

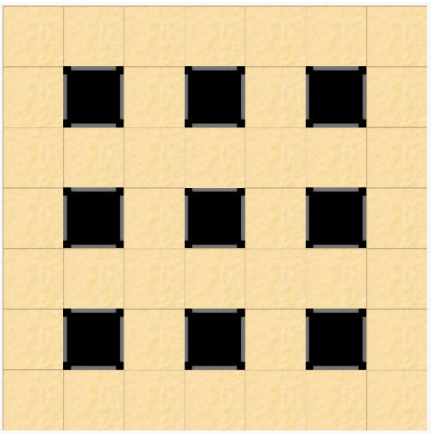
Uvažujme o klasickej kocke so šiestimi stranami. Mohli by sme ju použiť na generovanie náhodnej pozície na šachovnici s 6x6 políčkami? A čo šachovnica s 3x3 políčkami? Ako by sa zmenil spôsob generovania, keby sme použili mincu? Navrhnite takéto algoritmy generovania pozícií.

10.3. Pozorovanie náhodnosti

Pomocou algoritmu z predchádzajúcej úlohy a pomocou kocky vytvorte náhodné pozície na šachovnici. Výsledky zaznamenajte na šachovnicu. Dá sa vo výsledkoch pozorovať nejaká pravidelnosť?

10.4. Pripravenie náhodnej arény

Pripravte arénu. Vytvorte potomka triedy **`Arena`**, ktorú pomenujte napr. **`RandomArena`**. V konštruktore nastavte príslušnú veľkosť sveta. Odporúčame pravidelné rozloženie stien s jedným prázdnym poľom medzi stenami, ako je znázornené na nasledujúcom obrázku:



10.5. Pridanie generátora náhodných čísel

Pridajte referenčný atribút typu **`Random`** do triedy **`RandomArena`**. Nezabudnite, že trieda **`Random`** je definovaná v balíku **`java.util.Random`**. Inicializujte atribút v konštruktore.

10.6. Kontrola obsadenosti buniek

Do triedy **`RandomArena`** pridajte súkromnú metódu **`isCellFree()`**, ktorá bude požadovať dva parametre – stĺpec a riadok. Metóda vráti **`true`**, ak je bunka vo svete voľná (neobsahuje žiadnu inštanciu triedy **`Actor`**), inak vráti **`false`**.

10.7. Generovanie náhodných stien

Upravte konštruktor triedy **`RandomArena`** tak, aby náhodne generoval steny do tretiny počtu všetkých buniek arény.

10.8. Presun náhodnosti do predka

Metódy **`createRandomWall()`**, **`isCellFree()`** a atribút generátora (vrátane jeho inicializácie v konštruktore) presuňte do predka **`Arena`**. Nezabudnite presunúť aj riadky importu.

10.9. Zovšeobecnenie náhodného generovania

Do metódy **`createRandomWall()`** pridajte parameter typu **`Actor`** – bude to objekt, ktorý vložíme na náhodné súradnice. Zmeňte názov metódy (napr. **`insertActorRandomly()`**) a aktualizujte volanie metódy z triedy **`RandomArena`**.

10.10. Pridanie bonusov

Vytvorte triedu **`Bonus`** ako potomka triedy **`Actor`**. Vytvorte dve podtriedy triedy **`Bonus`** – triedu **`BonusFire`** a triedu **`BonusBomb`**. Nastavte pre tieto triedy vhodné obrázky.

10.11. Náhodné vytváranie bonusov

Upravte kód v metóde **`act()`** triedy **`BrickWall`**. Po jej zničení vygenerujte na jej mieste s pravdepodobnosťou 10 % bonusový oheň a s pravdepodobnosťou 10 % vygenerujte bonusovú bombu. V 80 % prípadov sa po zničení nič nevygeneruje.

10.12. Uplatnenie bonusu

Pripravte triedu **`Bonus`**. Vytvorte metódu **`protected applyYourself()`** bez návratovej hodnoty, ktorá má jeden parameter typu **`Player`**. Túto metódu nechajte v triede **`Bonus`** prázdnu. Potom v metóde **`act()`** definujte akciu, ktorá najprv zistí, či hráč stúpil na bonus (metóda **`(Player)this.getOneIntersectingObject(Player.class)`**), a ak áno, aplikuje ho (volaním virtuálnej metódy) a nakoniec odstráni bonus zo sveta.

10.13. Zvýšenie počtu bômb

Pridajte do triedy **`Player`** metódu **`public increaseBombCount()`** bez parametra, ktorá nemá návratovú hodnotu a ktorá zvýši počet bômb, ktoré môže hráč položiť, o jednu. Potom prepíšte metódu **`applyYourself()`** v triede **`BonusBomb`**. Uplatnenie bonusu znamená zvýšenie počtu bômb, ktoré má hráč k dispozícii, o jednu (volanie metódy **`increaseBombCount()`**).

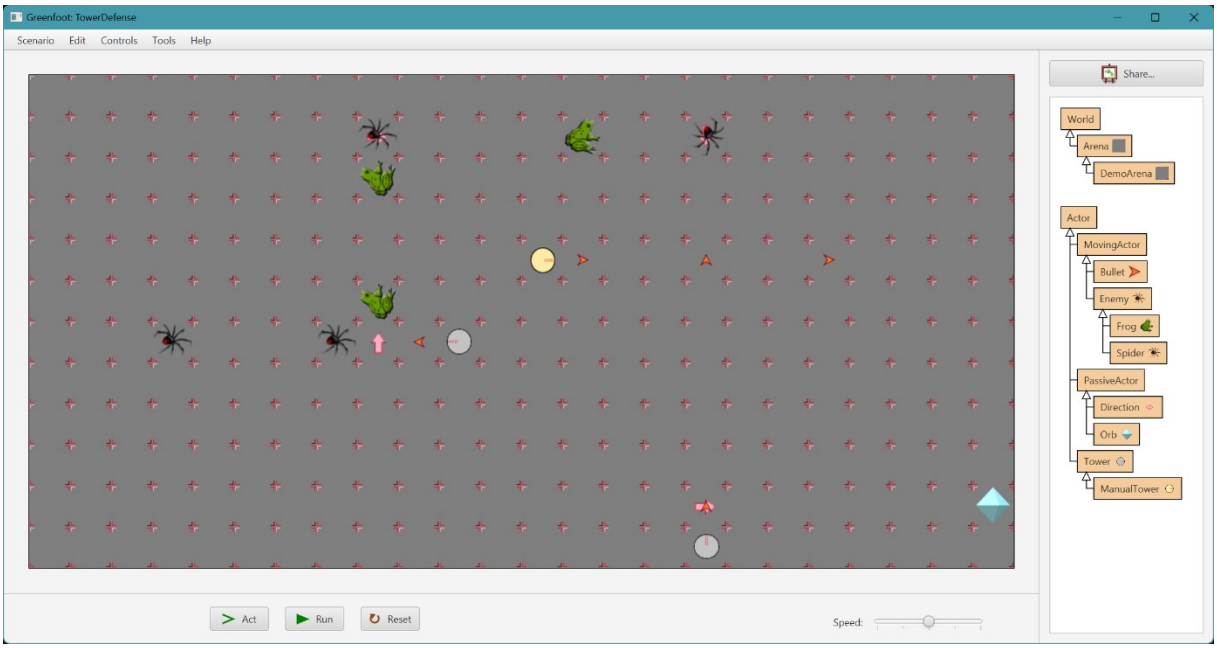
10.14. Zvýšenie sily bomby

Pridajte do triedy **`Player`** verejnú metódu **`increaseBombPower()`** bez parametra, ktorá nemá návratovú hodnotu a ktorá zvyšuje hodnotu atribútu **`bombPower`** o jedna. Potom prepíšte metódu **`applyYourself()`** v triede **`BonusFire`** a zvýšte silu bomby hráča o jedna.

3.2. Tower defense

V hrách typu tower defense (Obrana vežou) hráč používa veže, ktoré strieľajú určitý druh striel, aby zabránili nepriateľom dosiahnuť a zničiť cieľové miesto (označované angl. orb). Nepriatelia vždy sledujú rovnakú cestu, avšak ako hra pokračuje, nepriatelia sú silnejší a objavujú sa vo väčších skupinách. Hráč musí umiestniť veže na strategické miesta, aby ich zastavil v nadchádzajúcich vlnách. Existuje mnoho verzií hry, ktoré sa odohrávajú v rôznych svetoch s použitím rôznych entít (od balónov cez orkov, obranu pomocou veží podobných zvieratám až po magické bazény).

Tento projekt predstaví jeden typ veže, ktorú hráč môže alebo nemôže ovládať ručne. Nepriatelia budú rôznych typov s rozdielmi v ich HP a rýchlosti. Predstavený herný dizajn je ľahko rozšíriteľný, čo ponecháva dostatok priestoru pre kreativitu študentov, ako aj pre zadania učiteľa. Z tohto dôvodu sme sa v prvých kapitolách snažili minimalizovať aktivity hodnotiaceho typu. Okrem toho návrh ponecháva dostatok priestoru na prirodzené a jednoduché zavedenie tém mimo rozsahu ľahkého OOP (ako je polymorfizmus). Projekt v jeho konečnom stave počas hrania je znázornený na obrázku 3.



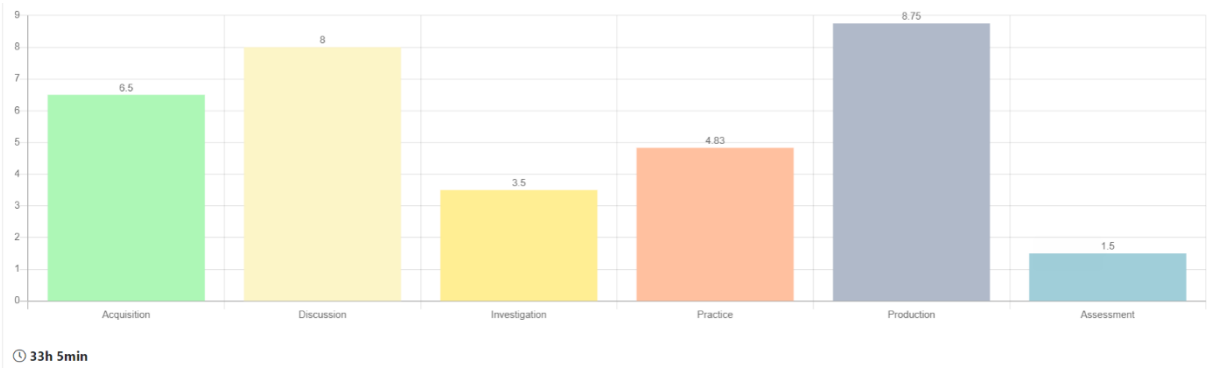
Obrázok 3: Prostredie Greenfoot s konečným stavom projektu Tower defense

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

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

3.2.1. Obsah a rozsah vzdelávacieho programu

Celkové pracovné zaťaženie študenta je 33 h 5 min a je rozdelené nasledujúco:



Obrázok 4: Pracovná záťaž študenta pri riešení projektu Tower defense

Konštruktívne smerovanie projektu je zosumarizované v nasledujúcej tabuľke:

Tabuľka 2: Konštruktívne smerovanie 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						

Pre detailnejší plán je potrebné prejsť na kapitolu 5.2.

3.2.2. Témy

Projekt tower defense je rozdelený do siedmich tém:

Aplikované témy z light OOP sú:

- triedy, objekty, inštancie,
- metódy, odovzdávanie argumentov metód,
- konštruktory,
- atribúty,
- statické premenné a metódy,
- zapúzdrenie,

- dedičnosť,
- abstraktné triedy,
- životný cyklus projektu.

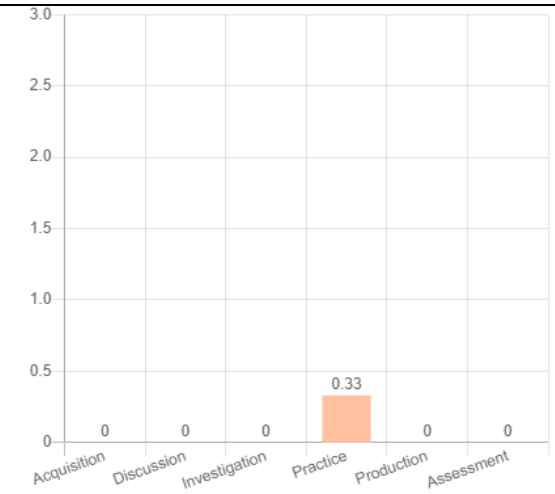
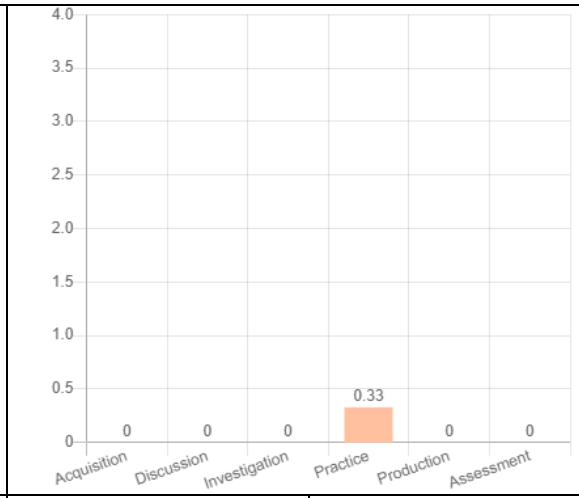
1. Úvod do prostredia Greenfoot

Téma je venovaná vytvoreniu projektu. Študenti budú schopní vytvoriť nový projekt v prostredí Greenfoot, vytvoriť triedu (ako potomka triedy **`Actor`**), vybrať obrázok pre novovytvorenú triedu, vytvoriť jej inštanciu a poslať jej správu.

Vytvorte nový projekt. Dajte mu správny názov (napr. **`TowerDefense`**) a uložte ho na správne miesto.

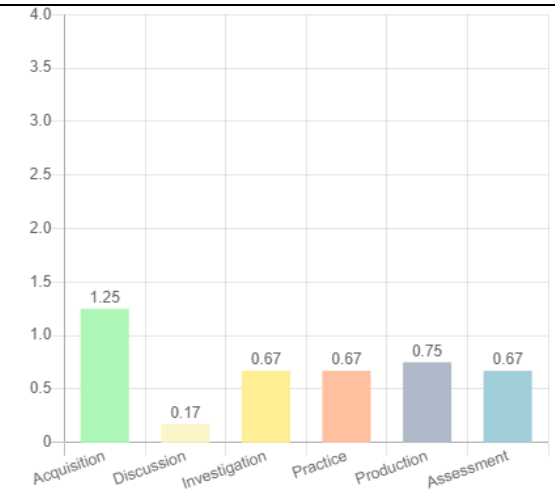
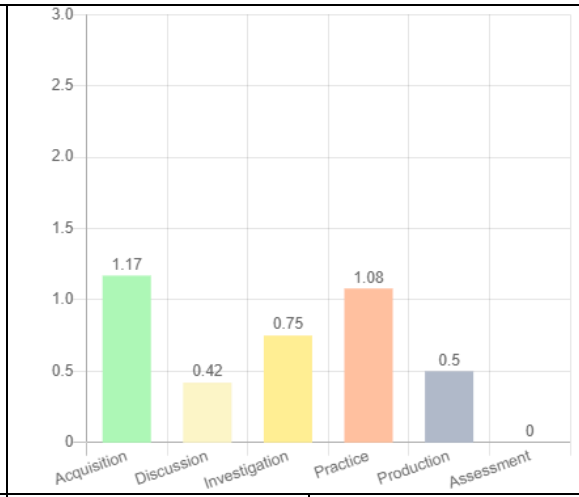
V tabuľke 3 je zhrnuté porovnanie pracovnej záťaže témy úvod do prostredia Greenfoot medzi projektmi Bomberman a Tower defense. V návrhu tém nie je žiadny rozdiel.

Tabuľka 3: Porovnanie pracovnej záťaže témy úvod do prostredia Greenfoot medzi projektmi Bomberman a Tower defense

	Bomberman		Tower defense
0h 20min		0h 20min	

V tabuľke 4 je zhrnuté porovnanie časové zaťaženie témy definícia tried medzi projektmi Bomberman a Tower defense. Pracovná záťaž projektu Tower defense je nižšia, viac prakticky orientovaná s dôrazom na skúmanie a precvičovanie.

Tabuľka 4: Porovnanie pracovnej záťaže témy definícia tried medzi projektmi Bomberman a Tower defense

	Bomberman		Tower defense
4h 10min		3h 55min	

1.1. Úloha 1.1 Identifikovanie objektov z projektu Bomberman.

1.2. Príprava sveta

Upravte zdrojový kód triedy **MyWorld** (dvakrát na ňu kliknite) a vytvorte svet s veľkosťou 24x12 buniek. Každá bunka by mala mať veľkosť 50 pixelov.

1.3. Príprava obrázku pre svet

Nájdite alebo vytvorte vhodný obrázok pre pozadie sveta. Môžete použiť buď pripravené obrázky (vyberte položku **Set image...** z kontextového menu triedy **MyWorld**, alebo vlastný obrázok (skopírujte obrázok do podadresára **images** v adresári projektu a vyberte ho rovnakým spôsobom, ako bolo popísané predtým).

Ako pozadie môžete použiť jediný obrázok, ktorý pokryje celú plochu sveta (vypočítajte potrebnú veľkosť obrázka vzhľadom na veľkosť sveta), alebo menší obrázok, ktorý sa bude opakovane kopírovať (použite štvorcový obrázok s veľkosťou jednej bunky).

1.4. Vytvorenie triedy Enemy

Vytvorte nepriateľa. Nepriateľ bude postupovať smerom k cieľovému miestu, aby ho poškodil a nakoniec zničil. Vytvorte nového potomka triedy **Actor** (z kontextového menu triedy **Actor** vyberte položku **New subclass...**). Dajte mu správne meno (**Enemy**) a obrázok.

1.5. Vytvorenie inštancie triedy Enemy

Vytvorte inštanciu triedy **Enemy** (z kontextového menu triedy **Enemy** vyberte položku **new Enemy()**, umiestnite inštanciu do sveta kliknutím ľavým tlačidlom myši na požadovanú pozíciu). Preskúmajte jej vnútorný stav (z kontextového menu vytvorenej inštancie vyberte položku **Inspect**).

Vytvorte ďalšiu inštanciu triedy **Enemy** a umiestnite ju na inú pozíciu. Porovnajte vnútorné stavy oboch vytvorených inštancií.

1.6. Odosielanie správ inštancii

Pošlite správu inštancii triedy **Enemy** (z kontextového menu vybranej inštancie vyberte položku zdedenú z triedy **Actor** a potom vyberte požadovanú položku). Čo sa stalo? Ako bol ovplyvnený vnútorný stav príslušnej inštancie?

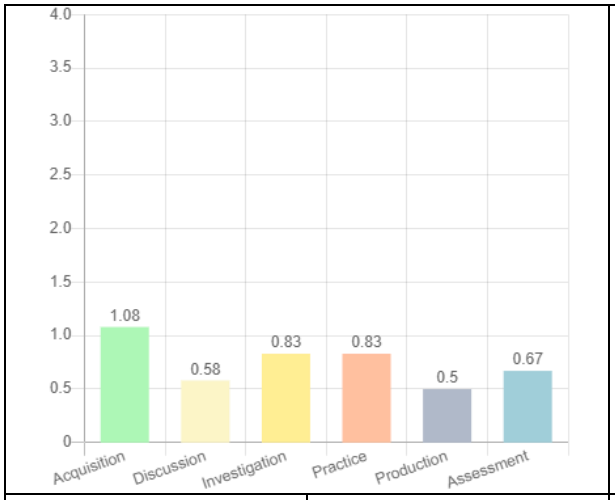
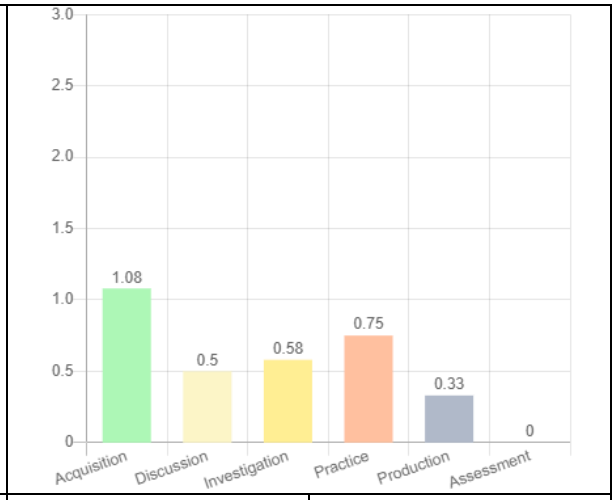
Pošlite správu inštancii triedy **Enemy**, aby sa presunula na pozíciu [12, 6] a bola otočená smerom nadol. Napíšte postupnosť odoslaných správ na papier.

2. Algoritmus, ovládacie prvky aplikácie, tvorba metód

Téma sa zaoberá základmi algoritmizácie a predstavuje základy práce s dokumentáciou. Študenti budú vedieť zavolať metódu v zdrojovom kóde, napísať a zobraziť dokumentáciu.

V tabuľke 5 je zhrnuté porovnanie pracovnej záťaže témy algoritmizácia medzi projektmi Bomberman a Tower defense. Projekt Tower defense je podobný projektu Bomberman, avšak s výrazne nižším počtom TLA typu skúmanie. Vidieť, že mnohé TLA sú rovnaké ako v projekte Bomberman. Je to preto, že v tomto stave projektov sa vo veľkej miere prekrýva to, čo sa s nimi dá robiť. Je možné nájsť inšpiráciu v úlohách v projekte Bomberman, ak bude potrebné posilniť časť skúmania učebných osnov. Pre potreby výučby light OOP s využitím projektu Tower defense však považujeme navrhnutý počet TLA úloh typu skúmanie za dostatočný.

Tabuľka 5: Porovnanie pracovnej záťaže témy algoritmizácia medzi projektmi Bomberman a Tower defense

			
Bomberman	4h 30min	Tower defense	3h 15min

2.1. Úloha 2.1 Napísanie jednoduchého algoritmu z projektu Bomberman.

2.2. Úloha 2.2 Napísanie všeobecnejšieho algoritmu z projektu Bomberman.

2.3. Zavolanie metódy

Do tela metódy **`act()`** pridajte príkaz, aby sa inštancia triedy **`Enemy`** posunula o dve bunky v aktuálnom smere. Potom vytvorte ďalšie inštancie triedy **`Enemy`** a zavolajte metódu na každej inštancii. Je toto správanie očakávané?

2.4. Pridanie dokumentácie

Pridajte dokumentačný komentár k metóde **`act()`**.

2.5. Pridanie ďalšej dokumentácie

Upravte dokumentačný komentár k triede **`Enemy`**. Pridajte verziu triedy a jej autora.

2.6. Úloha 2.7 Prečítanie si dokumentácie z projektu Bomberman

2.7. Úloha 2.9 Preskúmanie ovládacích prvkov aplikácie z projektu Bomberman

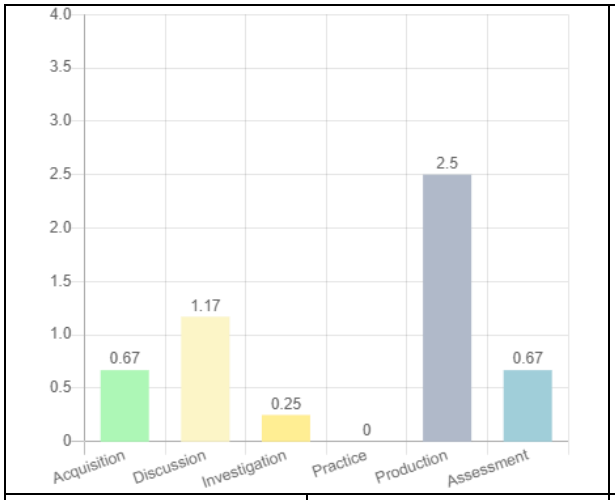
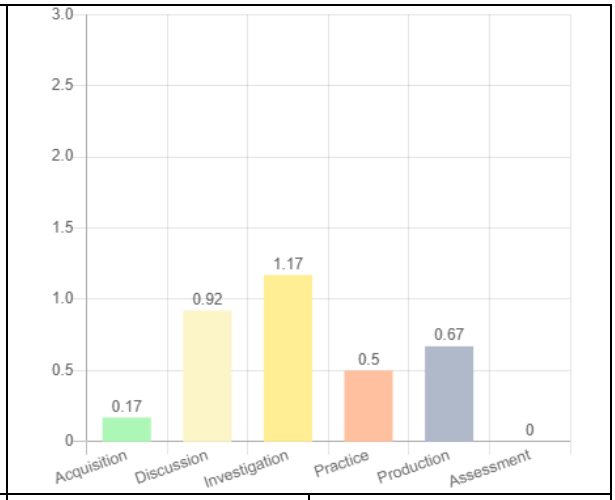
3. Vetvenie a riadenie nepriateľa

Téma zahŕňa neúplné a úplné vetvenie. Uvádzajú sa základy vnímania sveta (**`World`**) hercom (**`Actor`**). Študenti budú schopní písať kód pomocou podmienok.

Stav projektu v tejto kapitole otvára učiteľovi možnosti zadávať úlohy ako "použi inštancie triedy **`Direction`** a **`Orb`** na navigáciu nepriateľa (**`Enemy`**) tak, aby sa pohyboval v požadovanom smere", s obmedzujúcimi podmienkami ako použitie maximálneho počtu inštancií danej triedy alebo úlohy typu "predpovedaj pohyb v požadovanom nastavení sveta".

V tabuľke 6 je zhrnuté porovnanie pracovnej záťaže témy vetvenie medzi projektmi Bomberman a Tower defense. Medzi projektmi je zreteľný rozdiel. Tower defense rozdeľuje TLA typu tvorba na posilnenie skúmania a precvičovania. To umožňuje viac experimentovať a necháva otvorené dvere pre kreativitu študentov. Všimnite si nízky počet akvizičných TLA. Je to preto, že a) nie je zavedené viacnásobné vetvenie a b) je kladený dôraz na TLA typu skúmanie.

Tabuľka 6: Porovnanie pracovnej záťaže témy vetvenie medzi projektmi Bomberman a Tower defense

	Bomberman	5h 15min		Tower defense	3h 25min

3.1. Sledovanie stavu nepriateľa

Vytvorte inštanciu triedy **`Enemy`** a umiestnite ju do stredu hracej plochy. Otvorte okno s vnútorným stavom inštancie a umiestnite ho tak, aby bolo viditeľné počas behu aplikácie. Potom spustite aplikáciu a pozorujte, ako sa menia hodnoty atribútov **`x`**, **`y`** a **`rotation`** v inštancie triedy **`Enemy`**. Ako sa tieto hodnoty menia pri pohybe (hore, dole, vľavo a vpravo) a pri otáčaní?

3.2. Pridanie detekcie hraníc sveta

Do tela metódy **`act()`** pridajte kód na otočenie nepriateľa o 180° po dosiahnutí okraja sveta.

3.3. Pridanie tried Direction a Orb

Vytvorte dve nové triedy ako potomkov triedy **`Actor`**. Prvou triedou bude trieda **`Direction`** a druhou triedou bude **`Orb`**. V grafickom editore pripravte vhodné obrázky (maximálne 50x50 pixelov). Potom tieto obrázky priraďte k príslušným novovytvoreným triedam.

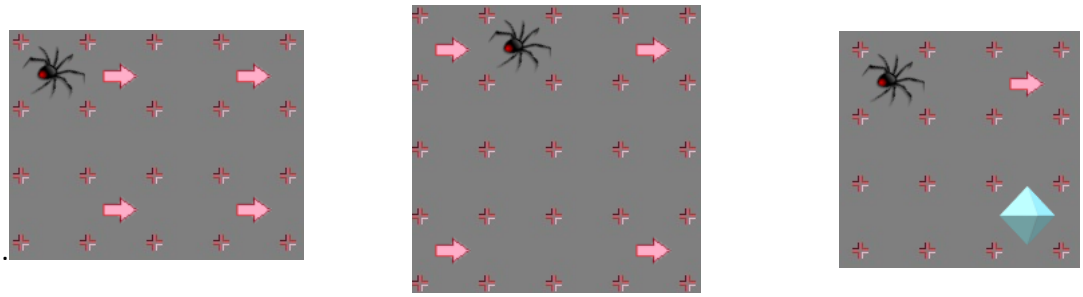
3.4. Pridanie detekcie kolízií

Do metódy **`act()`** triedy **`Enemy`** pridajte kód, ktorý zabezpečí, že:

- nepriateľ sa otočí o 90° v smere hodinových ručičiek, keď vstúpi do bunky, ktorá obsahuje inštanciu triedy **`Direction`**,
- nepriateľ sa otočí o 90° proti smeru hodinových ručičiek, keď vstúpi do bunky, ktorá obsahuje inštanciu triedy **`Orb`**.

3.5. Predvídanie pohybu nepriateľa na vlastnom usporiadaní

Pripravte si rôzne konfigurácie pre pohyb, pričom inšpiráciu nájdete na obrázkoch nižšie. Odhadnite, ako sa bude pohybovať nepriateľ. Spustite aplikáciu. Zodpovedá vaša predikcia tomu, čo pozorujete? Čo spôsobilo rozdiely v predikcii a skutočnosti?



a) b) c)

Obrázok 5: Konfigurácie vlastných nastavení inštancií na predpovedanie pohybu inštancie triedy Enemy

3.6. Predvídanie pohybu nepriateľa pri špecifickom usporiadaní

Pripravte si situáciu tak, ako je znázornená na obrázku nižšie. Odhadnite, ako sa bude pohybovať nepriateľ. Spustite aplikáciu. Zodpovedá vaša predikcia tomu, čo pozorujete? Čo spôsobilo rozdiely v predikcii a skutočnosti?

a) b) c)

Obrázok 6: Konfigurácie komplikovaných nastavení inštancií na predpovedanie pohybu inštancie triedy Enemy

3.7. Použitie úplného vetvenia pri detekcii kolízií

Z poslednej kapitoly vidíte, že problém nastáva, keď sa inštancia triedy **`Orb`** alebo **`Direction`** nachádza na okraji sveta alebo sú obe inštancie v tej istej bunke. Keď je vetvenie neúplné, dochádza ku kolíziám alebo opakovaným rotáciám. V podstate sú splnené viaceré podmienky súčasne. Preto je potrebné zmeniť kód metódy **`act()`** triedy **`Enemy`** tak, aby došlo len k jednej rotácii. Je potrebné použiť úplné vetvenie. Vytvorte kaskádu podmienok. Najdôležitejšou kontrolou (teda prvou) je detekcia hrán. Druhou najdôležitejšou kontrolou je kontrola dotyku inštancie triedy **`Direction`**. Poslednou je kontrola dotyku inštancie triedy **`Orb`**. Upravte metódu **`act()`** podľa týchto pravidiel.

3.8. Predvídanie pohybu nepriateľa na základe predchádzajúcich nastavení

Znovu prejdite úlohy 3.5 a 3.6. Čo sa zmenilo?

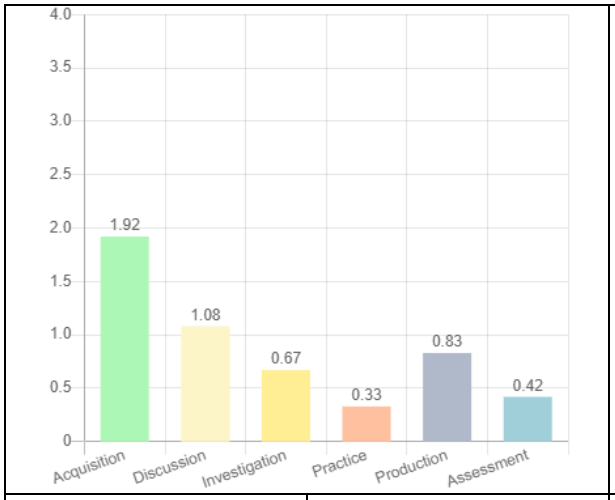
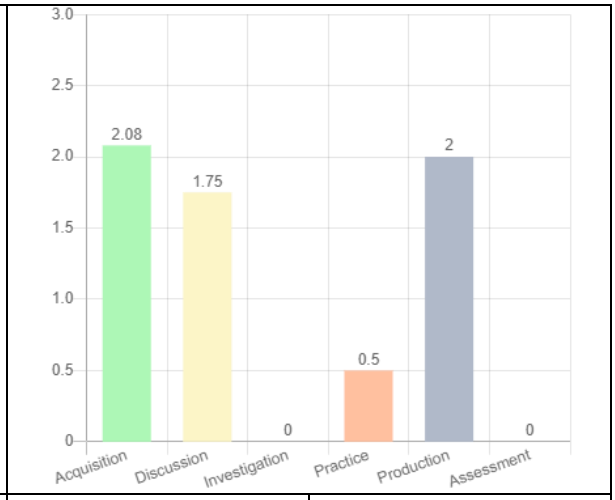
4. Premenné a výrazy

Táto kapitola sa venuje premenným a výrazom.

Stav projektu v tejto kapitole otvára podobné možnosti ako v predchádzajúcej kapitole. S trochou kreativity je možné pridať ďalšie triedy ako trieda **`Direction`**, ktoré spôsobia rôzne správanie pri zásahu inštanciou triedy **`Enemy`** (napr. teleporty, tunely atď.). O týchto triedach sa dá so študentmi diskutovať a príslušná implementácia sa môže zadať ako domáce zadanie.

V tabuľke 7 je zhrnuté porovnanie pracovnej záťaže témy premenné a výrazy medzi projektmi Bomberman a Tower defense. Pozrite si podobnosť návrhov predchádzajúcich a aktuálnych tém medzi oboma projektmi. Táto téma je viac orientovaná na tvorbu (podobne ako predchádzajúca téma v Bomberman) a naopak (všimnite si, že Bomberman je iný typ projektu, kde kreativita študenta bola výhodne využitá v tejto téme).

Tabuľka 7: Porovnanie pracovnej záťaže témy premenné a výrazy medzi projektmi Bomberman a Tower defense

	Bomberman	5h 15min		Tower defense	6h 20min

4.1. Otáčanie v smere

Zmeňte kód v metóde **`act()`** triedy **`Enemy`** tak, aby sa otáčal rovnakým smerom ako určuje inštancia triedy **`Direction`** (budú mať rovnakú rotáciu). Použite metódu **`getOneIntersectingObject(_cls_)`** na uloženie inštancie do správnej lokálnej premennej (**`Direction direction`** - budete musieť použiť pretypovanie, keďže návratová hodnota je typu **`Actor`**; pred volanie metódy **`getOneIntersectingObject`** napíšte **`(Direction)`**, k pretypovaniu sa dostaneme neskôr). Ak bola získaná nejaká inštancia príslušnej triedy, metódou **`getDirection()`** získajte natočenie pre nepriateľa (ak chcete, uložte ho) a potom ho nastavte pre nepriateľa (**`this`**) pomocou metódy **`setDirection(int)`**. Otestujte svoje riešenie.

4.2. Premenovanie triedy MyWorld na Arena

Dajte triede **`MyWorld`** lepšie meno. Premenujte ju na **`Arena`**. Nezabudnite na príslušné premenovanie konštruktora.

4.3. Vytvorenie usporiadania arény

Vytvorte vlastné rozloženie v triede **`Arena`**. Vyplňte konštruktor triedy. Pridajte jednu inštanciu triedy **`Enemy`**, jednu inštanciu triedy **`Orb`** a aspoň jednu inštanciu triedy **`Direction`**. Ak chcete pridať inštanciu potomka triedy **`Actor`**, môžete použiť nasledujúcu šablónu:

1. deklarujte a inicializujte premennú požadovaného typu (potomok triedy **`Actor`**),
   `Enemy e = new Enemy();`
2. nastavte vlastnosti pomocou vhodných metód,
   `e.setRotation(90);`
3. Vložte ju do sveta (**`Arena`**) pomocou metódy **`addObject(Actor)`**.
   `this.addObject(e, 6, 0);`

Otestujte svoje riešenie.

4.4. Identifikácia problému s pohybom a návrh riešenia

Zistite, čo spôsobuje problémy s pohybom. Ako sa dajú tieto problémy vyriešiť?

`Nepriateľ` *sa v súčasnosti pohybuje v jednom kroku 2 bunkami, čo spôsobuje problémy s pohybom. Rýchlosť nepriateľa môžeme modelovať inak. Inštancia triedy* ***`Enemy`*** *sa bude vždy pohybovať len 1*

*bunkou v jednom kroku. Zavedieme však oneskorenie pohybu – inštancia triedy **`Enemy`** sa bude pohybovať až po danom počte (**`delay`**) zavolaní metódy **`act()`**.*

4.5. Atribút moveDelay triedy Enemy

Pridajte do triedy **`Enemy`** nový atribút typu **`int`** s názvom **`moveDelay`**. Vytvorte parametrický konštruktor s parametrom na inicializáciu tohto atribútu. Inicializujte atribút pomocou parametra. Podľa týchto zmien upravte kód v triede **`Arena`**.

4.6. Pohyb nepriateľov s dodržaním oneskorenia

Aktualizujte metódu **`act()`** triedy **`Enemy`** tak, aby sa pohybovala po **`moveDelay`**-tom volaní metódy **`act()`**. Zaveďte nový atribút **`nextMoveCounter`**. Inicializujte ho na **`0`**. Upravte metódu **`act()`** tak, aby volala **`this.move(1)`** len vtedy, ak **`nextMoveCounter`** dosiahne hodnotu **`0`**. Po vykonaní pohybu nastavte **`nextMoveCounter`** na hodnotu **`moveDelay`**. Ak sa inštancia triedy **`Enemy`** nemohla pohnúť (pretože **`nextMoveCounter`** nedosiahol **`0`**), znížte **`nextMoveCounter`** o **`1`**.

4.7. Parametrický konštruktor triedy Direction

Pridajte parametrický konštruktor do triedy **`Direction`** s jediným parametrom **`rotation`** typu **`int`**. Vytváranú inštanciu v tele konštruktora otočte podľa parametra. Podľa uvedených zmien upravte kód v triede **`Arena`**.

4.8. Preťaženie konštruktorov v triede Direction

Preťažte konštruktory v triede **`Direction`** pridaním neparametrického konštruktora. V tele neparametrického konštruktora zavolajte parametrický konštruktor s parametrom **`direction`** rovným **`0`**. Podľa toho upravte kód v triede **`Arena`** – kde je to možné, volajte neparametrickú verziu konštruktora triedy **`Direction`**.

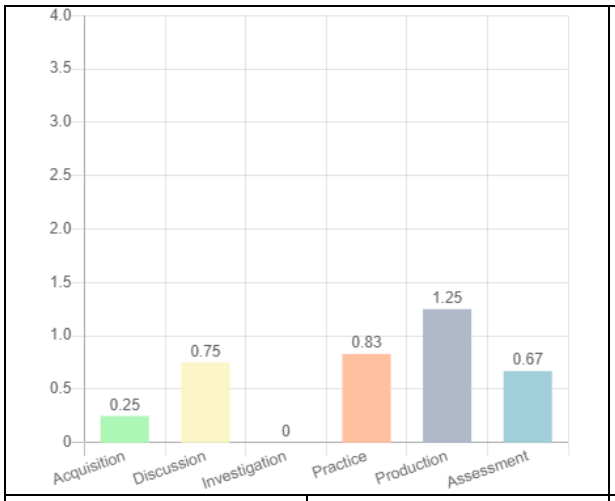
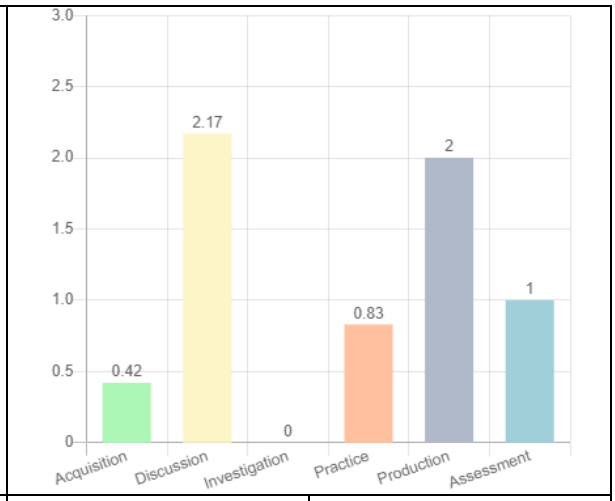
5. Asociácia

Najdôležitejšia téma tohto projektu je zameraná na asociáciu. V rámci diskusie sa zisťuje, ako môže spolupráca rôznych objektov priniesť zložité správanie, aj keď sú kódy v objektoch ľahko pochopiteľné a udržiavateľné. Na znázornenie spolupráce objektov a rozdelenia algoritmu medzi spolupracujúcimi objektmi sa používajú sekvenčné diagramy UML. Tento diagram sa dá zostaviť počas diskusie s triedou.

Projekt možno prehlásiť za dokončený po tejto téme. Nasledujúce kapitoly predstavujú väčšiu variabilitu aplikácie so zameraním na výhody OOP pri jeho správnom použití.

V tabuľke 8 je zhrnuté porovnanie pracovnej záťaže témy asociácia medzi projektmi Bomberman a Tower defense. Pochopenie asociácie považujeme za najdôležitejšiu kompetenciu pri využití tohto projektu na výučbu OOP. Preto sme výrazne posilnili TLA typu tvorba a diskusia. Všimnite si, že posilnená je aj TLA typu hodnotenie. TLA tohto typu sú navrhnuté tak, aby sa využili predtým vykonané TLA v trochu inom kontexte.

Tabuľka 8: Porovnanie pracovnej záťaže témy asociácia medzi projektmi Bomberman a Tower defense

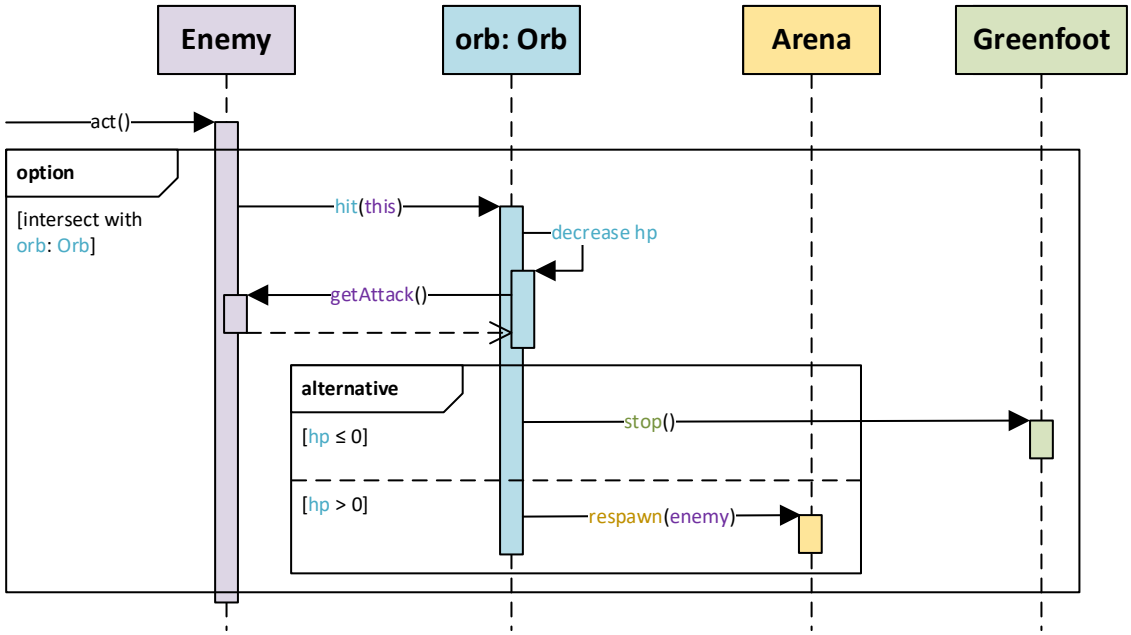
	Bomberman	3h 45min		Tower defense	6h 25min

5.1. Diskusia o tom, čo by sa malo stať, keď nepriateľ dosiahne cieľové miesto

Keď sa nepriateľ dostane ku cieľovému miestu, toto miesto si zníži HP. Ak je HP = 0, hra sa skončí, inak sa nepriateľ respawnuje v aréne.

5.2. Diskusia o tom, ako by mala inštancia triedy Enemy interagovať s príslušnými objektmi pomocou správ pri zásahu inštancie triedy Orb

Algoritmus sa rozloží medzi spolupracujúce objekty.



Obrázok 7: UML sekvenčný diagram interakcie inštancie triedy Enemy s inými objektmi pri dosiahnutí inštancie triedy Orb

5.3. Atribúty attack triedy Enemy a hp triedy Orb

Pridajte nový atribút typu **`int`** s názvom **`attack`** do triedy **`Enemy`**. Pridajte parameter do konštruktora na inicializáciu tohto atribútu. Inicializujte atribút pomocou parametra.

Pridajte nový atribút typu **int** s názvom **hp** do triedy **Orb**. Pridajte parametrický konštruktor s parametrom na inicializáciu tohto atribútu. Inicializujte atribút pomocou parametra.

Príslušne upravte kód v triede **Arena**.

5.4. Získanie hodnoty atribútu attack triedy Enemy

Vytvorte getter (metódy na získanie hodnoty atribútu) atribútu **attack** v triede **Enemy**.

5.5. Vytvorenie a testovanie metódy respawn(Enemy) triedy Arena

Pridajte metódu **respawn** bez návratovej hodnoty a s jediným parametrom typu **Enemy** do triedy **Arena**. V metóde nastavte polohu a rotáciu nepriateľa na rovnaké hodnoty ako pri vytvorení v konštruktore.

Otestujte metódu. Po vytvorení inštancie **Arena** nespúšťajte aplikáciu. Namiesto toho presuňte myšou inštanciu triedy **Enemy**. Potom vyvolajte kontextové menu inštancie triedy **Arena** (musíte kliknúť pravým tlačidlom myši v aréne, kde nie je inštancia inej triedy) a vyberte položku **respawn**. Ak chcete vyplniť parameter, uistite sa, že aplikácia je pozastavená a súbor s parametrom je aktívny (s blikajúcim kurzorom vo vnútri). Ak áno, kliknite ľavým tlačidlom myši na inštanciu triedy **Enemy**. Sledujte, aký výraz sa vytvoril v okne. Potom kliknite na tlačidlo OK a sledujte, čo sa stane.

5.6. Vytvorenie a testovanie metódy hit(Enemy) triedy Orb

Pridajte metódu **hit** bez návratovej hodnoty s jediným parametrom typu **Enemy** do triedy **Orb**. Nechajte telo prázdne.

Otestujte volanie metódy. Použite podobné kroky ako vyššie, avšak vyvolajte kontextové menu inštancie triedy **Orb**. Sledujte, aký výraz sa vytvoril v okne.

5.7. Zavolanie metódy hit(Enemy) triedy Orb z triedy Enemy

Zmeňte kód v metóde **act()** triedy **Enemy** tak, aby sa metóda **hit()** zavolala, keď inštancia triedy **Enemy** zasiahne inštanciu triedy **Orb**.

Odstráňte staré kódy, ktoré spôsobovali otáčanie nepriateľa pri zásahu orbu a ktoré spôsobovali, že sa nepriateľ odrážal od okraja sveta.

5.8. Implementácia metódy hit(Enemy) triedy Orb

Implementujte telo metódy **Orb.hit(Enemy)** s ohľadom na analýzu vykonanú v úlohe 5.2. Otestujte svoju aplikáciu.

5.9. Pridanie tried Bullet a Tower

Pridajte triedy **Bullet** a **Tower**. Použite rovnaké princípy ako v úlohe 3.3.

5.10. Diskusia o tom, ako sa má pohybovať inštancia triedy Bullet a čo sa má stať, keď dosiahne inštanciu triedy Enemy alebo okraj arény.

Guľka (Bullet) by sa mala pohybovať, kým nedosiahne nepriateľa alebo okraj sveta. Guľka nemení smer pohybu. Rýchlosť guľky možno riadiť pomocou rovnakého mechanizmu ako v úlohe 4.6.

5.11. Implementácia pohybu inštancie triedy Bullet

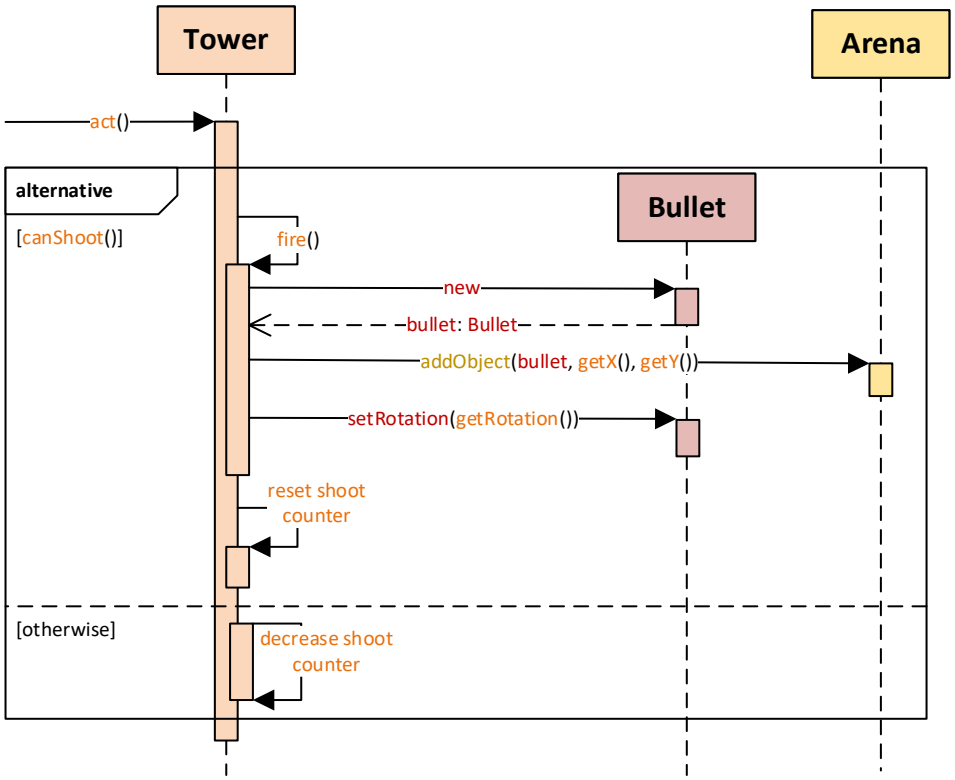
Použite vedomosti nadobudnuté v úlohách 3.2, 3.4 a 4.6. Pridajte do kódu, kde by malo dôjsť k interakcii s inštanciou triedy **Enemy** dokumentačné komentáre.

5.12. Diskusia o tom, ako inštancia triedy Tower vystrelí inštanciu triedy Bullet

*Nezabudnite, že inštancia **Tower** by nemala strieľať pri každom volaní metódy **act()**. Inšpirujte sa mechanizmom použitým v časti 4.6. Rozdeľte príslušné kroky do metód triedy **Tower**.*

5.13. Diskusia o tom, ako by mala inštancia triedy Tower komunikovať s príslušnými objektmi pomocou správ pri streľbe

Použite podobné princípy ako v 5.2.



Obrázok 8: UML sekvenčný diagram interakcie inštancie triedy Tower s inými objektmi pri vytváraní inštancií triedy Bullet

5.14. Implementácia streľby inštancie triedy Tower

Postupujte podľa výstupu úlohy 5.13:

- najskôr pripravte potrebné atribúty a konštruktor,
- potom vytvorte metódy **`boolean canShoot()`** and **`void fire()`** v triede **`Tower`** (prvá nech vráti hodnotu **`false`**, druhá nech nerobí nič aby ste ich mohli použiť v metóde **`act()`**),
- nakoniec implementujte telo metódy **`act()`**.

Implementujte metódu **`canShoot()`** tak, aby vrátila **`true`** ak počítadlo streľby dosiahne hodnotu **`0`**.

Implementujte metódu **`fire()`** nasledovne:

- zavolajte konštruktor triedy **`Bullet`** a vytvorenú inštanciu uložte do lokálnej premennej (**`Bullet bullet`**),
- pridajte vytvorenú guľku do arény na rovnakých súradniciach aké má inštancia triedy **`Tower`**,
- nastavte guľke rovnakú rotáciu akú má inštancia triedy **`Tower`**.

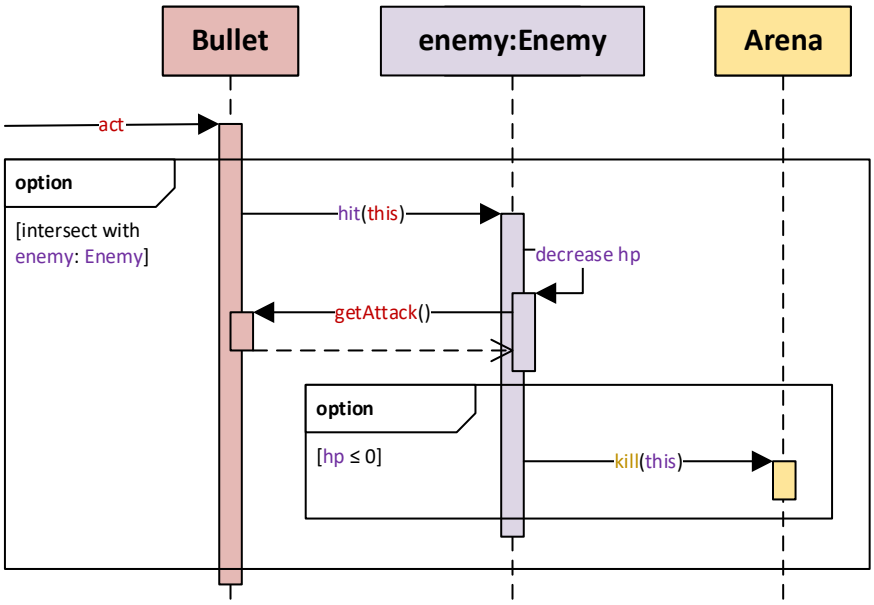
Otestujte svoje riešenie.

5.15. Veže v aréne

Preťažte konštruktor triedy **`Tower`** tak, aby obsahoval aj parameter **`int rotation`** (analogicky ako v 4.8). Aktualizujte konštruktor triedy **`Arena`** tak, aby sa inštancie triedy **`Tower`** umiestnili podľa potreby. Použite správny konštruktor triedy **`Tower`**.

5.16. Diskusia o tom, ako by mala inštancia triedy Bullet komunikovať s príslušnými objektmi pomocou správ

Použite podobné princípy ako v 5.2.



Obrázok 9: UML sekvenčný diagram interakcie inštancie triedy Bullet s inými objektmi pri zásahu inštancie triedy Enemy

5.17. Implementácia zásahu inštancie triedy Enemy inštanciou triedy Bullet

Postupujte podľa výstupu úlohy 5.16:

- najskôr pripravte atribúty a metódy (analogicky podľa úloh 5.3, 5.4, 5.5 a 5.6),
- potom zavolajte metódu **`hit(Bullet)`** triedy **`Enemy`** z inštancie triedy **`Bullet`** (analogicky ku 5.7) kde bol zanechaný komentár v úlohe 5.11,
- napokon implementujte metódu **`hit(Bullet)`** v triede **`Enemy`** (analogicky ku 5.8).

Otestujte svoje riešenie.

5.18. Vznik nepriateľov a koniec hry

Na vyvolanie vytvárania nepriateľov použite metódu **`act()`** triedy **`Arena`**. Správne implementujte oneskorenie medzi vytvorením nepriateľov. Proces vytvárania (vytvoriť inštanciu triedy **`Enemy`**, priradiť jej vlastnosti, pridať ju do arény) implementujte v metóde **`spawn()`** triedy **`Arena`**. Počítanie vytvorených inštancií triedy **`Enemy`** implementujte v atribúte triedy **`Arena`** (inicializované na **`0`**, zvýšenie pri vytvorení, zníženie pri zabití). Zmeňte metódu kill(Enemy) v triede **`Arena`** – ak je posledný nepriateľ zabitý, hráč vyhral hru – zastavte Greenfoot a vypíšte príslušnú správu na obrazovku.

6. Dedičnosť

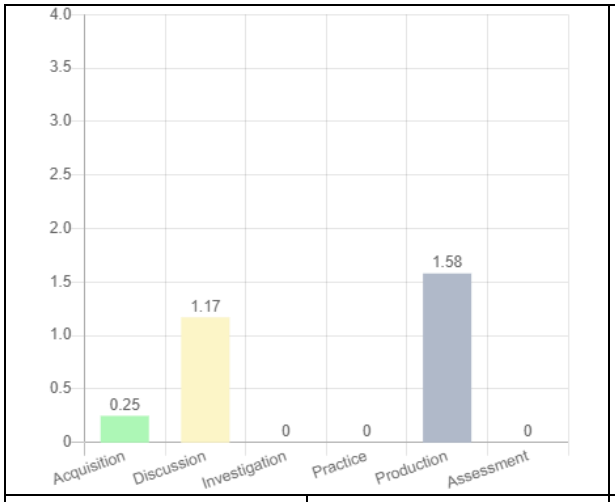
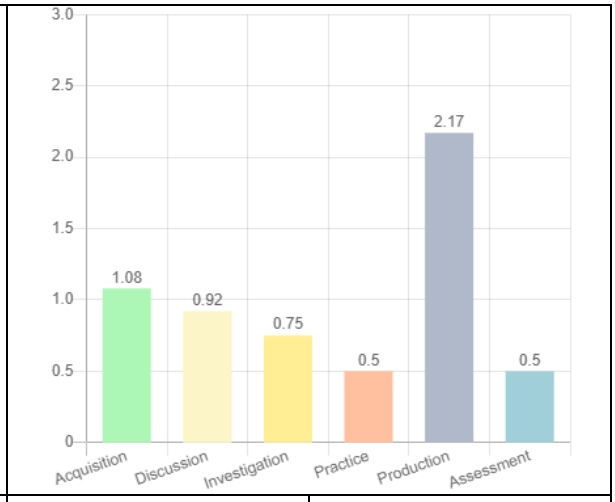
Táto téma uvádza do projektu rôznorodosť prostredníctvom dedičnosti. Zavedieme Liskov princíp substitúcie, aby sme ukázali výhody OOP. Dôrazne odporúčame nechať študentov experimentovať a vymýšľať vlastných potomkov nepriateľov a arén. Keďže tieto budú mať spoločné rozhranie, bude jednoduché dať všetko dokopy. Podobne ako v téme 4 môže byť vytvorených veľa domácich úloh.

Ako už bolo spomenuté, projekt sa môže nachádzať v dokončenej fáze po predchádzajúcej téme. Preto v prípade potreby môže učiteľ túto tému upraviť tak, aby ukázal výhody dedičnosti a s ňou spojenú

univerzálnosť len na tu navrhovaných hierarchiách tried **Arena** alebo **Enemy**. To povedie k skráteniu počtu hodín spojených s touto témou.

V tabuľke 9 je zhrnuté porovnanie pracovnej záťaže témy dedičnosť medzi projektmi Bomberman a Tower defense. Návrh pri experimentovaní sa premieta do väčšieho počtu TLA typu skúmanie, precvičovanie a tvorba. V tejto téme je zahrnutá rozsiahlejšia teoretická časť so zameraním na princíp Liskovej substitúcie.

Tabuľka 9: Porovnanie pracovnej záťaže témy dedičnosť medzi projektmi Bomberman a Tower defense

	Bomberman	3h 0min		Tower defense	5h 55min

6.1. Identifikácia spoločných vlastností tried Orb a Direction

*Inštancie tried **Orb** a **Direction** nepotrebujú špecifickú implementáciu v metóde **act()**. Reagujú len na správy. Môžeme zaviesť spoločného predka, vďaka ktorému bude metóda **act()** prázdna a potomkovia budú prehľadnejší.*

6.2. Pridanie triedy PassiveActor ako predka tried Orb a Direction

Vytvorte novú triedu **PassiveActor**. Zmeňte kódy tried **Orb** a **Direction** tak, aby boli potomkami triedy **PassiveActor**. Odstráňte metódu **act()** z tried **Orb** a **Direction** – je zdedená z triedy **PassiveActor**.

6.3. Zavedenie abstraktnosti pre triedu PassiveActor

Pri vytváraní metód triedy **PassiveActor** sa stretávame s metódami, ktoré nie je možné implementovať v spoločnej triede, a preto musíme ich implementáciu prenechať potomkom. Ako príklad uvažujme triedu **Tvar** s potomkami **ObdĺžnikObdĺžnik** a **Trojuholník**. Každý tvar bude mať implementované metódy obvodu a obsahu, ktoré však nemožno implementovať v spoločnej triede. Ak označíme metódu triedy ako abstraktnú, v podstate tým hovoríme, že ju bude implementovať potomok. Trieda obsahujúca abstraktnú metódu musí byť abstraktná. Preto do hlavičky triedy pridáme slovo **abstract**.

6.4. Identifikácia spoločných vlastností tried Bullet a Enemy

*Inštancie tried **Bullet** a **Enemy** sa počas svojho života správajú podobne. Pohybujú sa rovnakým spôsobom a potom reagujú na okolie. Môžeme zaviesť spoločného predka, ktorý bude implementovať metódu **act()**, aby sa pohybovali rovnakým spôsobom, a potomkovia sa zamerajú na svoj špecifický účel.*

6.5. Pridanie abstraktnej triedy MovingActor ako predka tried Bullet a Enemy

Použite podobný postup ako v prípade 6.2.

6.6. Identifikácia atribútov tried Bullet a Enemy potrebných na pohyb

Preskúmajte metódy **`act()`** príslušných tried. Identifikujte atribúty **`moveDelay`** a **`nextMoveCounter`**. Všimnite si, že kód metódy **`act()`** zodpovedný za pohyb je rovnaký.

6.7. Presun kódu zodpovedného za pohyb do triedy MovingActor

- Presuňte atribúty identifikované v časti 6.6 z potomkov **`Bullet`** a **`Enemy`** do triedy **`MovingActor`** (odstráňte ich z potomkov).
- Pridajte parametrický konštruktor do triedy **`MovingActor`** na inicializáciu týchto atribútov.
- Zavolajte konštruktor predka so správnymi parametrami z potomkov **`Bullet`** a **`Enemy`**.
- Presuňte kód zodpovedný za pohyb v metóde **`act()`** z potomkov **`Bullet`** a **`Enemy`** do predka **`MovingActor`** (odstráňte kód z potomkov, zvyšok metódy tam ponechajte).
- Zavolajte metódu predka **`act()`** ako prvý riadok v metóde **`act()`** v potomkoch **`Bullet`** a **`Enemy`**.

6.8. Vytvorenie vlastných nepriateľov

- Pridajte potomka triedy **`Enemy`** , ktorý bude predstavovať rôznych nepriateľov (napr. **`Frog`** a **`Spider`**). Uistite sa, že obrázky nepresahujú veľkosť bunky.
- Pridajte do tried bezparametrický konštruktor, ktorý bude volať konštruktor predka (triedy **`Enemy`**) s parametrami špecifickými pre každý druh nepriateľa.
- Odstráňte metódu **`act()`** v potomkoch (alebo pridajte volanie **`super`**).

6.9. Vytváranie vlastných nepriateľov

Aktualizujte metódu **`spawn()`** triedy **`Arena`**. Vytvorte inštanciu triedy **`Frog`** alebo **`Spider`** a uložte ju do premennej typu **`Enemy`**. Na rozhodnutie, ktorá inštancia sa má vytvoriť, použite ľubovoľný druh rozhodnutia (môže byť náhodné, môže byť presne spočítané atď.). Dbajte na to, aby sa v aplikácii nemenili žiadne iné kódy.

6.10. Diskusia o hierarchii arén

Diskutujte a navrhnite hierarchiu tried **`Arena`**. Potomkovia triedy **`Arena`** budú zodpovední za vlastné rozloženie – pozíciu inštancií tried **`Orb`** a **`Direction`** a veľkosť arény. Tieto úlohy sa budú vykonávať v konštruktore potomkov. Čo bude potrebné odovzdať do parametrov konštruktora triedy predka (**`Arena`**)? Nezabudnite, že všetko ostatné (vytváranie, znovu vytvorenie a zabíjanie nepriateľov) sa bude vykonávať v triede **`Arena`**.

Mali by ste identifikovať potrebu nastaviť a uložiť polohu a natočenie pri vytváraní nepriateľov. To sa vykoná pomocou atribútov a príslušných parametrov konštruktora. Okrem toho by mal konštruktor akceptovať aj rozmery plochy.

6.11. Vytvorenie univerzálnej arény

Vytvorte atribúty **`int spawnPositionX`** , **`int spawnPositionY`** a **`int spawnRotation`** a použite ich v metódach **`spawn()`** a **`respawn(Enemy)`**. Pridajte parametre do konštruktora triedy **`Arena`** na ich inicializáciu.

Pridajte ďalšie dva parametre do konštruktora triedy **`Arena`** – **`int width`** a **`int height`**. Tieto parametre odovzdajte konštruktoru predka.

Všimnite si, že **`Arena`** nemôže byť automaticky vytvorená v prostredí **`Greenfoot`**, , pretože potrebuje parametre pre konštruktor. Preto nastavte triedu **`Arena`** ako abstraktnú.

6.12. Vytvorenie triedy DemoArena

Pridajte potomka **DemoArena** triedy **Arena**. Zavolajte konštruktor predka triedy **DemoArena** s parametrami, ktoré zabezpečia vytvorenie arény rovnakých rozmerov a vytváranie nepriateľov rovnakým spôsobom ako predtým.

Presuňte kód zodpovedný za rozloženie inštancií tried **Direction**, **Orb** a **Tower** z konštruktora triedy **Arena** do konštruktora triedy **DemoArena**.

Vytvorte inštanciu triedy **DemoArena** – z kontextového menu triedy **DemoArena** vyberte položku **new DemoArena()**.

6.13. Vytvorenie vlastných arén

Pomocou podobného prístupu ako v časti 6.12 vytvorte ďalších zaujímavých potomkov triedy **Arena**. O svoj kód sa môžete podeliť s ostatnými študentmi vo vašej skupine.

7. Zapúzdrenie

Posledná téma je zameraná na správne používanie súkromných metód, metód tried a premenných tried. Použitie metód tried a premenných možno nahradiť (netriednymi) atribútmi a metódami v triede **Arena** (v kontexte nášho projektu je prítomná len jedna inštancia triedy **Arena**). To umožní implementovať úlohy z tejto témy, ale s využitím už známych konceptov.

Zapuzdrenie sme už použili v projekte Bomberman. Tu sa použijú statické atribúty a statické metódy. Doteraz boli všetky atribúty a metódy viazané na inštanciu triedy. Predstavme si situáciu, že by sme chceli počítať vystrelené náboje v triede **Bullet**. Počet vystrelených nábojov je spoločný pre všetky inštancie triedy **Bullet** a nezávisí od konkrétnej inštancie. Takýto atribút sa nazýva statický a keďže je spoločný pre triedu, pristupuje sa k nemu cez názov triedy, napr. **Bullet.count**. Podobne môžeme vytvoriť napríklad metódu, ktorá vráti počet nábojov. Opäť bude spoločná pre všetky inštancie triedy. Bude teda statická. Statické atribúty a statické metódy majú vo svojej deklarácii kľúčové slovo **static**. Všimnite si, že môžu existovať, aj keď neexistuje inštancia triedy. Statické metódy sa inicializujú v samotnej definícii.

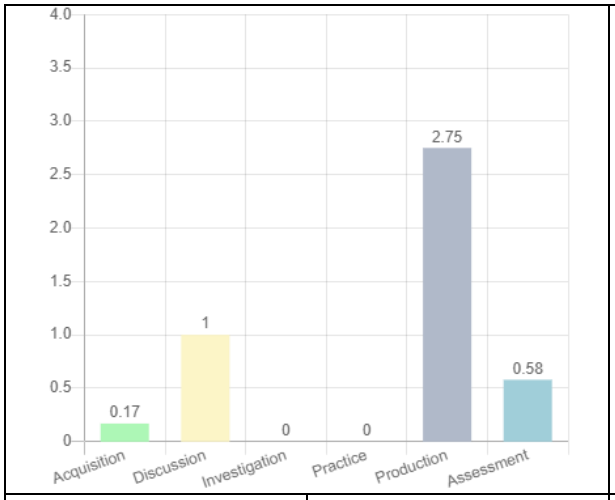
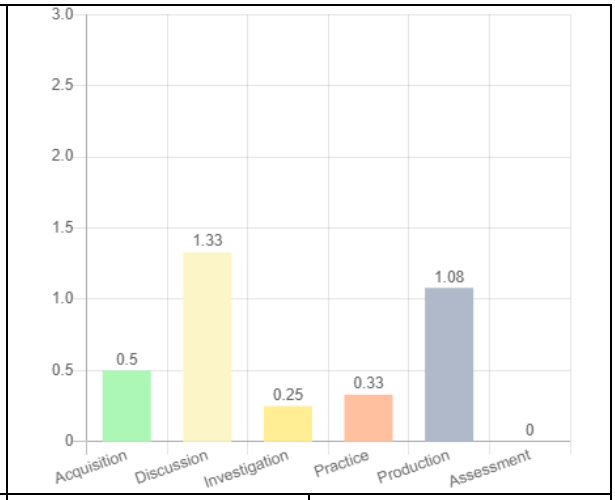
```
static int countOfBullets = 0;

static int countOfBullets(){

...
```

V tabuľke 10 je zhrnuté porovnanie pracovnej záťaže témy zapuzdrenie medzi projektmi Bomberman a Tower defense. Podobne ako v predchádzajúcej téme je tvorba rovnomernejšie rozdelená medzi ostatné typy TLA. Vyššie množstvo akvizičných TLA pochádza zo zavedenia metód tried a premenných. Ako bolo naznačené, existuje možnosť vyhnúť sa týmto konceptom, čo povedie k redukcii týchto TLA a čo vo výsledku povedie k podobnému návrhu ako pri projekte Bomberman.

Tabuľka 10: Porovnanie pracovnej záťaže témy zapúzdrenie medzi projektmi Bomberman a Tower defense

	Bomberman	4h 30min		Tower defense	3h 30min

7.1. Vytvorenie triedy ManualTower

Vytvorte triedu **`ManualTower`** ako potomka triedy **`Tower`**. Pripravte obrázky tejto triedy, keď je pod kontrolou hráča a keď nie je. Pridajte dva konštruktory v rovnakom duchu ako konštruktory predka a zabezpečte volanie konštruktora predka. Nech metóda **`act()`** volá rodičovskú metódu **`act()`**.

Pridajte inštancie tejto triedy do rozloženia vybranej arény.

7.2. Zmena ovládania manuálne ovládanej veže

Pridajte atribút **`boolean isManuallyControlled`** a inicializujte ho na **`false`**. Vytvorte metódu **`void changeControl(boolean)`** v nej zmeňte atribút a príslušne aj obrázok.

7.3. Vyvolanie zmeny manuálneho ovládania

Manuálne vyvolajte zmenu spôsobu ovládania vybranej inštancie triedy **`ManualTower`**. Sledujte zmeny vnútorného stavu podobne ako v časti 1.6.

7.4. Spracovanie ovládania používateľa

Vytvorte súkromnú metódu **`void processUserControl()`**. V nej sa najskôr zistí, či bolo kliknuté na túto inštanciu. Ak áno, zmení sa na ručné ovládanie. Potom implementujte samotné ručné ovládanie. Otestujte, či je inštancia v manuálnom režime, a ak áno, získajte objekt **`MouseInfo`**. Ak bol objekt získaný, otočte inštnciu triedy **`MaualTower`** smerom k pozícii kurzora myši.

Pred odovzdaním vykonania metódy **`act()`** predka (**`super.act()`**) zavolajte metódu **`processUserControl()`** z metódy **`act()`**. V metóde skontrolujte, či bolo na túto inštanciu kliknuté. Ak áno, zavolajte metódu **`changeControl`** so správnou hodnotou.

Otestujte svoje riešenie opätovným vykonaním 7.3.

7.5. Identifikácia problému s používateľským ovládaním a návrh riešenia

Identifikujte, čo je problematické pri používateľskom ovládaní. Ako možno tieto problémy vyriešiť?

V súčasnosti nie je možné zrušiť výber veže. K aktuálne ovládanej inštancii by mala existovať evidencia, ktorá sa deaktivuje, keď sa vyberie nejaká iná inštancia. Pridajte evidenciu ručne ovládanej veže.

7.6. Pridanie evidencie ručne riadenej veže

Pridajte do triedy **`ManualTower`** atribút typu **`ManualTower`**, ktorý predstavuje odkaz na ručne ovládanú inštanciu, a inicializujte ho na **`null`**. Tento atribút musí byť spoločný pre všetky inštancie triedy ManualTower, musí byť preto označený kľúčovým slovom **`static`**. Skontrolujte vnútorný stav triedy (z kontextového menu triedy **`ManualTower`** vyberte položku **`Inspect`**). Čo bolo pridané?

7.7. Zmena manuálne ovládanej veže z centralizovaného miesta

Pridajte metódu **`changeControlledInstance`** na zmenu ručne ovládanej veže ako metódu triedy **`ManualTower`** (teda definovanú s kľúčovým slovom **`static`**). Parametrom metódy by mal byť odkaz na inštanciu triedy **`ManualTower`**, ktorá bude ručne ovládaná.

Ak sa parameter metódy líši od ručne ovládanej inštancie (v statickom atribúte), použite metódu **`changeControll`** so správnymi parametrami. Najprv pomocou **`changeControll`** nastavte pôvodnú ručne ovládanú inštanciu na hodnotu **`false`**. Potom zmeňte statický atribút triedy **`ManualTower`** ručne ovládanej inštancie na parameter novej metódy (t. j. novej ručne ovládanej veže). Nakoniec použite metódu **`changeControll`** na nastavenie novej ručne ovládanej inštancie na hodnotu **`true`**. Nezabudnite ošetriť prípadné **`null`** odkazy. Môže totiž nastať situácia, že nebude existovať žiadna ručne ovládaná veža.

Otestujte svoje riešenie. Z kontextového menu triedy **`Tower`** vyberte položku s novovytvorenou metódou. Na vyplnenie parametra môžete použiť rovnaký princíp ako v časti 5.5.

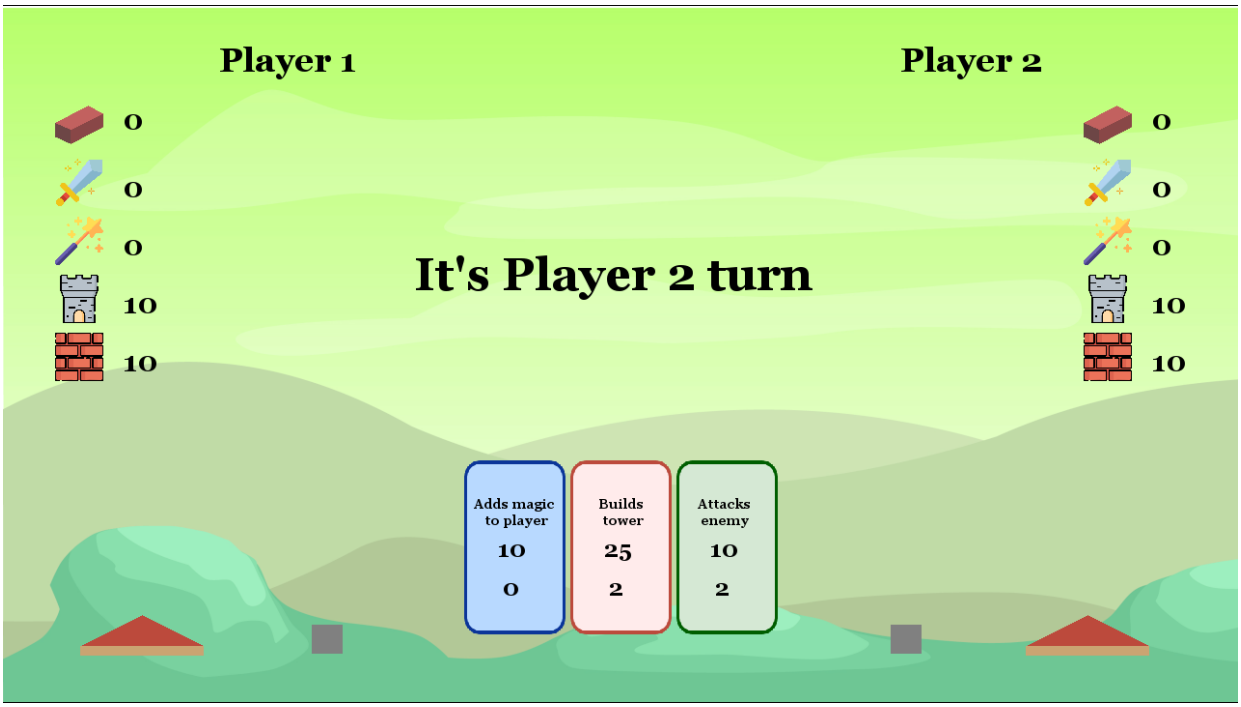
*Mali by ste si všimnúť, že novovytvorená metóda triedy sa nevolá dôsledne, inštancie triedy **`ManualTower`** obchádzajú evidenciu pri spracovaní vstupu, čo spôsobuje problémy.*

7.8. Vyvolanie zmeny ručne ovládanej veže

Vyvolajte **`changeControlledInstance(ManualTower)`** triedy **`ManualTower`** z príslušných miest. Nakoniec urobte metódu **`changeControl(boolean)`** triedy **`ManualTower`** súkromnou. Pozorujte zmeny v rozhraní inštancie triedy **`ManualTower`** podobne ako v 1.6.

3.3. Projekt Mravce

Mravce (angl. Ants) je kartová hra pre dvoch hráčov. Každý hráč má svoju vlastnú vežu a stenu a zdroje, ako sú tehly, meče a kúzla. Jeden ťah hry pozostáva z akcie, pri ktorej hra ponúkne hráčovi 3 náhodne karty a on si jednu vyberie. Existujú tri typy kariet - karty stavieb, bojové karty a magické karty. Stavebné karty možno použiť na zväčšenie vlastnej veže alebo hradby, bojové karty na útok na nepriateľského hráča a magické karty na zvýšenie počtu vlastných surovín alebo ukradnutie nepriateľských.



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

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

3.3.1. Témy

Projekt mravce je rozdelený do 6 tém:

Aplikované témy z light OOP sú:

- triedy, objekty, inštancie,
- metódy, odovzdávanie argumentov metód,
- konštruktory,
- atribúty,
- statické premenné a metódy,

- zapúzdrenie.

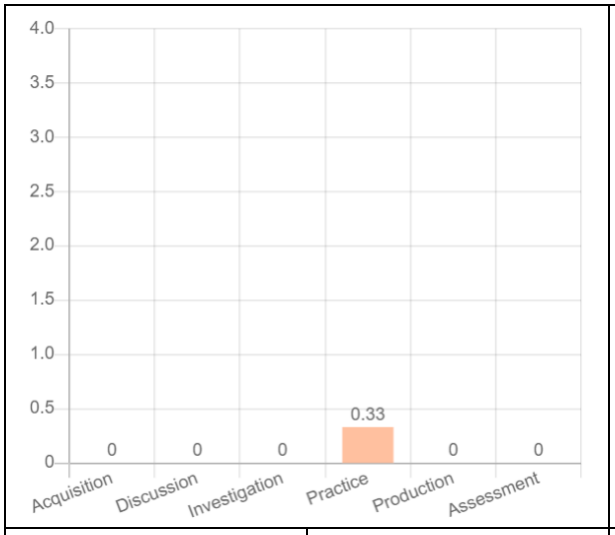
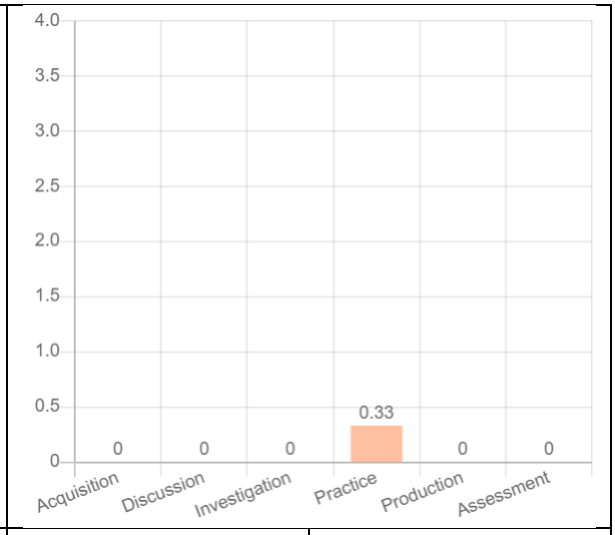
1. Prostredie Greenfoot, definícia triedy, základná práca s triedou

Táto téma je venovaná tvorbe projektov. Študenti budú schopní vytvoriť nový projekt v prostredí Greenfoot, vytvoriť triedu (ako potomka triedy **`Actor`**), vybrať obrázok pozadia, vytvoriť inštanciu vytvorenej triedy a poslať jej správu.

Vytvorte nový projekt. Dajte mu vhodný názov (napr. **`Ants`**) a uložte ho na vhodné miesto.

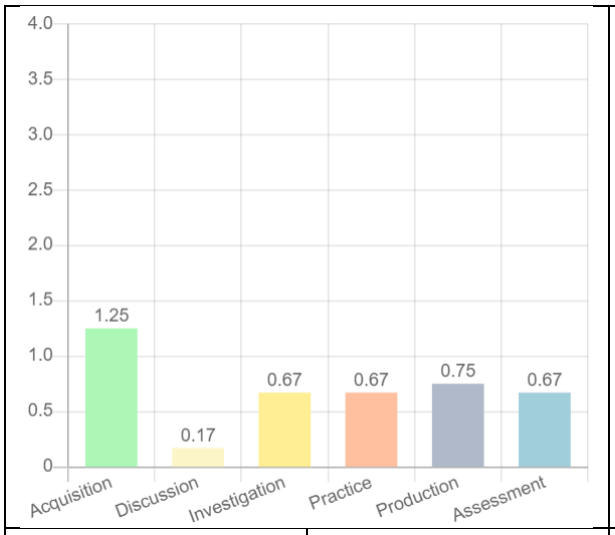
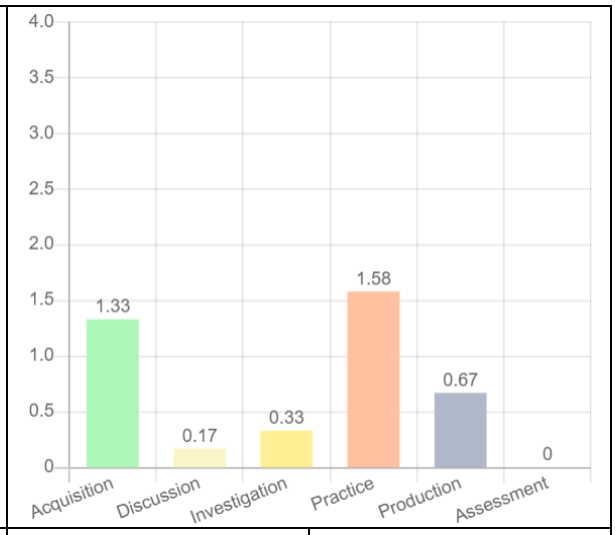
V tabuľke 11 je zhrnuté porovnanie pracovnej záťaže témy úvod do prostredia Greenfoot medzi projektmi Bomberman a Mravce. Celková pracovná záťaž oboch projektov je v tejto téme rovnaká.

Tabuľka 11: Porovnanie pracovnej záťaže témy úvod do prostredia Greenfoot medzi projektmi Bomberman a Mravce

			
Bomberman	0h 20min	Ants	0h 20min

V tabuľke 12 je zhrnuté porovnanie pracovnej záťaže témy definícia triedy a základná práca s triedami medzi projektmi Bomberman a Mravce. Celková pracovná záťaž oboch projektov je podobná. Hlavný rozdiel je v precvičovaní a hodnotení. Ako však bude uvedené v ďalšej časti, niektoré praktické úlohy môžu byť zadané aj ako hodnotenia, čo môže projekt Mravce ešte viac vyvážiť.

Tabuľka 12: Porovnanie pracovnej záťaže témy definícia triedy a základná práca s triedami medzi projektmi Bomberman a Mravce

			
Bomberman	4h 10min	Ants	4h 05min

1.1. *Úvod do prostredia Greenfoot*

Vytvorte nový projekt v prosterdí Greenfoot a oboznámte študentov so základnými prvkami, používateľským rozhraním atď. Počiatočný stav úložiska obsahuje aj súbory, ktoré môžete v projekte použiť.

1.2. *Vytvorenie triedy Wall*

Vytvorte triedu **`Wall`** ako potomka triedy **`Actor`**. Predstavte študentom pojmy ako triedy, hierarchia tried, inštancie atď.

1.3. *Vytvorenie triedy Tower*

Podobne ako v predchádzajúcej časti vytvorte triedu **`Tower`**. Môžete to nechať na študentov ako individuálnu úlohu.

1.4. *Definovanie atribútu/poľa triedy*

Predstavte študentom pojmy pole/atribút, primitívne typy atď. Pokúste sa identifikovať polia vo vašich triedach (usmernite študentov konkrétne na výšku (**`height`**)) a definujte ich v triede **`Wall`**.

1.5. *Priradenie hodnoty atribútu/poli*

Diskutujte o hodnotách a priradeniach polí a priraďte k atribútu **`height`** triedy **`Wall`** hodnotu **`10`**.

1.6. *Definovanie a priradenie hodnoty atribútu/poľa pre triedu Tower*

Ako individuálnu prácu nechajte študentov zopakovať to isté pre triedu **`Tower`**.

1.7. *Konštruktory tried*

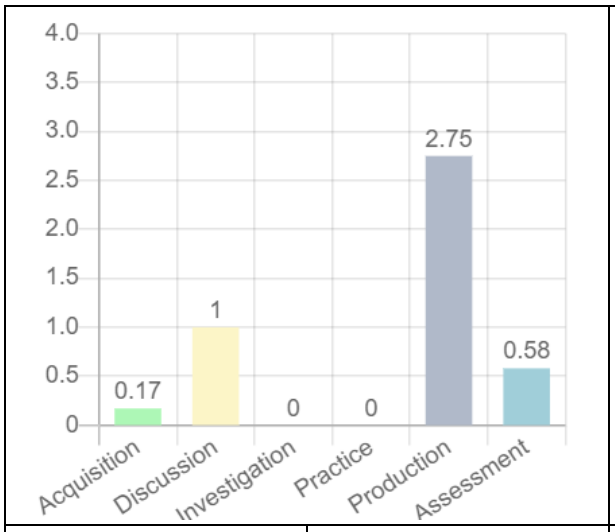
Diskutujte o inštanciách, triedach a konštruktoroch. Presuňte priradenie hodnoty do konštruktora.

2. Zapúzdrenie, kompozícia, metódy

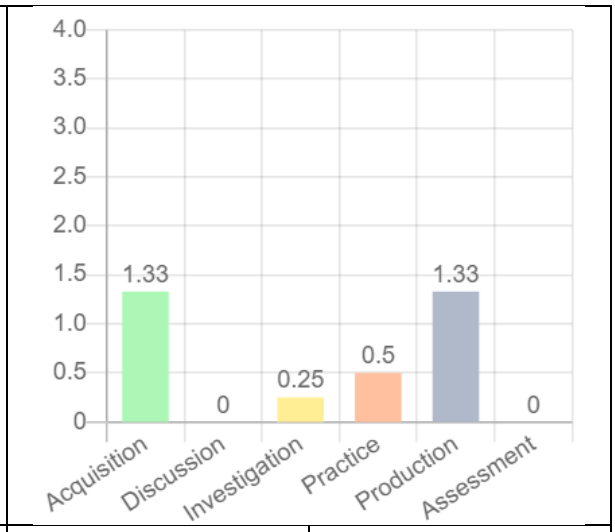
V tejto téme sú uvedené základné princípy OOP, konkrétne pojmy ako zapuzdrenie, kompozícia a metódy. Študenti sa dozvedia, ako a prečo by mali byť polia/atribúty zapuzdrené a nemali by byť poskytované ako verejné, ako sa objekty skladajú z iných objektov a ako vytvárať metódy a volať ich. Študenti v tejto časti vytvoria inštanciu triedy **`Player`** a dajú mu polia typu objekt, konkrétne **`Wall`** a **`Tower`**.

Tabuľka 13 ukazuje rozdiely v dvoch podobných témach - zapuzdrenie z projektu Bomberman a zapuzdrenie, kompozícia a metódy v projekte Mravce. Ako vidieť, projekt Mravce sa viac zameriava na akvizíciu a na druhej strane Bomberman sa viac zameriava na tvorbu a diskusiu. Je to spôsobené tým, že táto téma v projekte Mravce pozostáva z viacerých oblastí ako zapuzdrenie, preto je potrebné aj viac akvizície. Celková pracovná záťaž je v projekte Mravce o hodinu kratšia ako v Bombermanovi, pretože tieto pojmy sú vysvetlené plytšie, ale v ďalších častiach sa tieto vedomosti upevnia.

Tabuľka 13: Porovnanie pracovnej záťaže témy zapúzdrenie, kompozícia, metódy v projekte Mravce a podobnej témy v projekte Bomberman - zapúzdrenie



Bomberman	4h 30min



Ants	3h 25min

2.1. Definovanie metód

Porozprávajte sa o zapuzdrení a vysvetlite študentom, prečo nie je dobré vytvárať napríklad výšku (**`height`**) steny ako verejnú vlastnosť a radšej ju zapuzdriť v príslušnej metóde (getter). Potom vytvorte metódu **`getHeight()`** v triede **`Wall`**.

2.2. Definovanie metód s parametrami

Vysvetlite parametre metódy a definujte metódu **`increaseHeight`** v triede **`Wall`**, ktorá zvýši výšku steny o zadané číslo.

2.3. Zopakovanie tvorby metód pre triedu Tower

Ako samostatnú prácu môžete dať študentom úlohu, aby si predtým ukázané prístupy zopakovali aj na triede **`Tower`**.

2.4. Kompozícia objektov

Vysvetlite študentom, čo je to kompozícia a prečo je potrebné mať polia s prvkami typu inštancia triedy. Pokúste sa identifikovať takéto typy pre každého hráča v tejto hre. Potom vytvorte triedu **`Player`** pridajte mu atribút typu **`Wall`** a **`Tower`**.

2.5. Vytvorenie inštancie triedy

Vysvetlite konštruktor a vytvorte inštancie triedy **`Wall`** a **`Tower`** v konštruktore triedy **`Player`**.

2.6. Volanie metód inštancie

Vysvetlite študentom, ako môžete volať metódy vytvorených inštancií. Potom sa ich pokúste zapuzdriť tak, aby ich bolo možné volať z prostredia mimo triedy **`Player`** prostredníctvom inštancie tejto triedy. Vytvorte metódy **`getWallHeight`**, **`getTowerHeight`**, **`increaseWallHeight`** a **`increaseTowerHeight`** v triede **`Player`**.

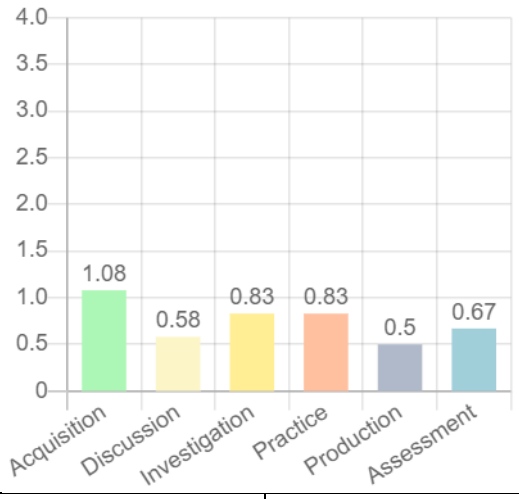
3. Konštruktory, zložitejšie volania metód (práca s grafikou v prostredí Greenfoot)

Táto téma je zameraná na volania metód pomocou volania objektov prostredia Greenfoot na kreslenie inštancií. Študenti budú kresliť inštancie tried **`Wall`**, **`Tower`** a **`Player`**.

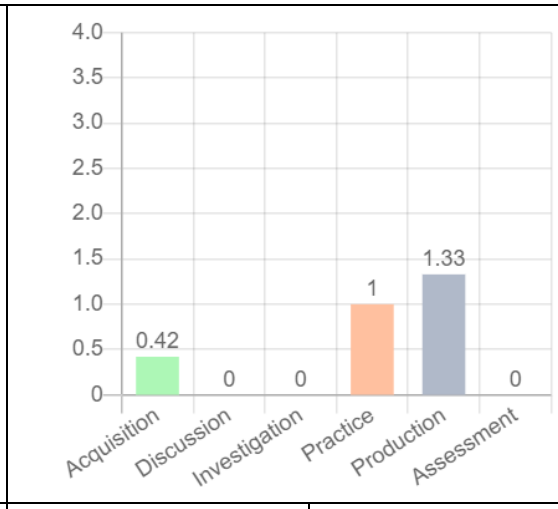
V tabuľke 14 je uvedené porovnanie tejto témy v projekte Mravce a najpodobnejšej témy v projekte Bomberman, ktorou je algoritmus. Všimnite si, že tieto témy sú trochu odlišné, pretože naša ukazuje

základnú prácu s konštruktormi a predstavuje grafiku v prostredí Greenfoot a v projekte Bomberman je viac zameraná na algoritmus. Preto aj z tohto dôvodu možno poukázať na rozdiely.

Tabuľka 14: Porovnanie pracovnej záťaže témy konštruktory, zložitejšie volania metód (práca s grafikou v prostredí Greenfoot) v projekte Mravce a podobnej témy v projekte Bomberman - algoritmus, ovládacie prvky aplikácie, vytváranie metód



Bomberman	4h 30min	Ants	2h 45min



3.1. Kreslenie objektov v prostredí Greenfoot – Wall

Predstavte študentom konštanty v Jave – definujte **`wallSizeX`** a **`wallSizeY`** ako **`static final`** (kľúčové slovo **`final`** zaistí, že sa hodnota nedá zmeniť, teda že sa jedná o konštantu) atribúty (konštanty) s hodnotami 32 a 3. Ku konštante sa pristupuje ako ku statickej premennej prostredníctvom názvu triedy, napr. **`Wall.wallSizeX.`** Potom implementujte funkciu **`draw`** v triede **`Wall`**, kde sa vytvorí nový obrázok s danou veľkosťou a vyplní sa ako obdĺžnik.

3.2. Kreslenie objektov v prostredí Greenfoot – Tower

To isté sa opakuje aj v prípade triedy **`Tower`**. Avšak je to komplikovanejšie, pretože veža sa skladá aj zo strechy, ktorá je realizovaná ako polygón. Polygón vyžaduje pole bodov. Ak chcete, môžete študentom podať stručné vysvetlenie polí, hoci polia nie sú súčasťou tejto časti. Porozprávajte sa so žiakmi o analógii pojmov pole a zoznam.

3.3. Definovanie ďalších vlastností hráča

Pokúste sa identifikovať ďalšie vlastnosti hráča – konkrétne meno hráča a počet tehál, mečov a kúziel. Potom tieto vlastnosti implementujte ako atribúty v triede **`Player`** a inicializujte ich v konštruktore.

3.4. Vykreslenie hráča

Poslednou úlohou v tejto téme je vykreslenie triedy **`Player`**. Musíte nakresliť ikony jednotlivých zdrojov a počet týchto zdrojov a tiež meno hráča.

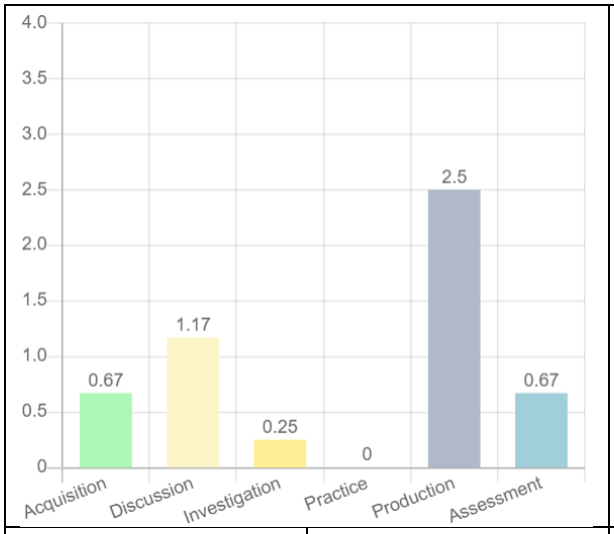
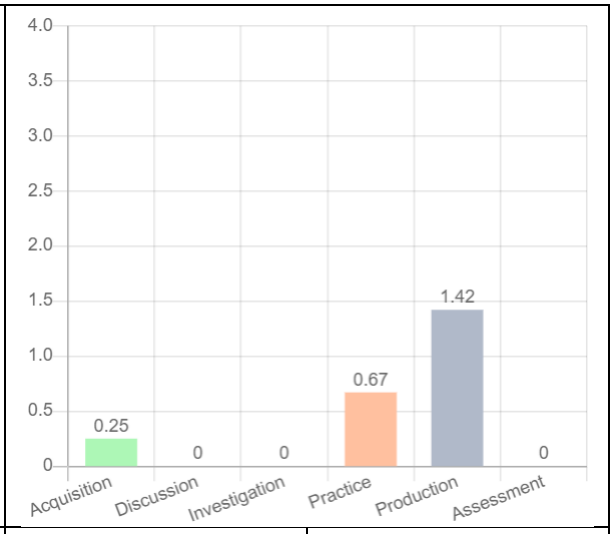
4. Vetvenie, podmienené vykonávanie

Táto téma sa zameriava na vetvenie programu a podmienené vykonávanie častí našej hry. V tejto hre je to potrebné v niekoľkých prípadoch, ako napríklad vykreslenie prvého hráča v ľavej časti obrazovky a druhého v pravej atď.

Tabuľka 15 ukazuje rozdiely medzi podobnými témami v hrách Bomberman a Mravce. Ako je vidieť, projekt Bomberman kladie väčší dôraz na diskusiu ako projekt Mravce. Aj celková pracovná záťaž v hodinách je približne polovičná v porovnaní s projektom Bomberman. Je to spôsobené tým, že vetvenie

je v tomto projekte rozdelené do viacerých častí - táto časť je skôr úvodom a základným použitím vetvenia.

Tabuľka 15 Porovnanie pracovnej záťaže témy vetvenie, podmienené vykonávanie medzi projektmi Bomberman a Mravce

	Bomberman	5h 15min		Ants	2h 20min

4.1. Vytvorenie triedy Hra

Predtým, ako začnete do vášho kódu vkladať logiku vetvenia, musíte vytvoriť triedu **`Game`**, ktorá bude obsahovať oboch hráčov a v budúcnosti bude spravovať logiku hry, prepínanie ťahov, vykonávanie kariet atď. Zatiaľ tam vložte atribúty pre dvoch hráčov a vytvorte ich inštancie v konštruktore triedy **`Game`**.

4.2. Vetvenie, podmienené vykonávanie kódu – hráči sú zobrazení na príslušných stranách herného plánu

Teraz by sa mal zaviesť nový atribút pre triedu **`Player`** – informácia o tom, či sa má hráč vykresliť na ľavej alebo pravej strane obrazovky. Túto informáciu použite na nastavenie príslušných vlastností hráča – pozícia steny, veže, hud (angl. head-up display) a mena. Tento posun na základe pozície hráča by sa potom mal pridať do metódy **`redraw`**.

4.3. Pridávanie inštancií do sveta

Ďalšia úloha vytvorí počiatočné zobrazenie inštancií triedy **`Player`** pre hru a vytvorí inštanciu hry vo svete.

4.4. Pridanie inštancií do sveta 2

Aby sa inštancia triedy **`Player`** správne vykreslila pri vytvorení inštancie triedy **`Game`**, je potrebné pridať do sveta **`Wall`** a **`Tower`** a zavolať metódu **`redraw`** v metóde **`act`**. Tu si môžete vysvetliť vykonávanie hernej slučky (metóda **`act`**).

4.5. Podmienenie vykonania kódu len raz

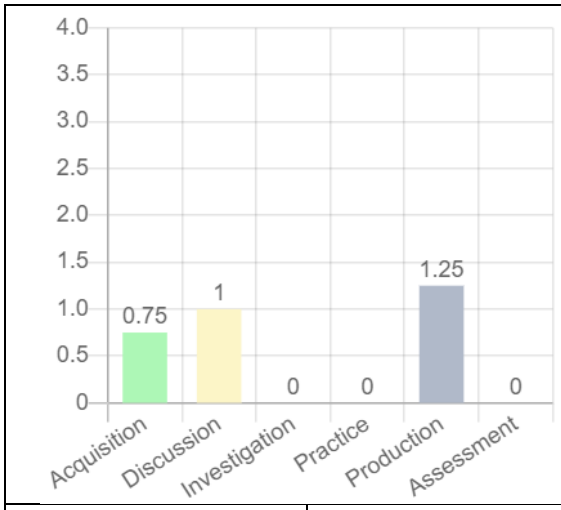
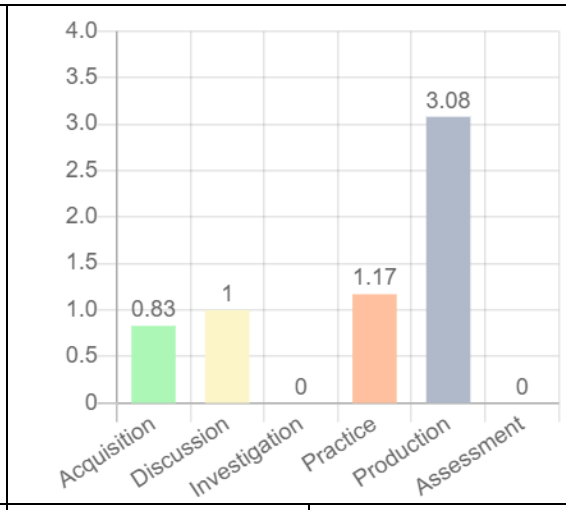
Pri pokuse o spustenie hry po poslednej úlohe môžete naraziť na problém - objekty sa do sveta pridávajú viackrát za sekundu. Môžete dať študentom úlohu na odstránenie tohto problému alebo ho odstrániť spolu s nimi. Jedným z riešení je zavedenie nového atribútu typu **`boolean`**, ktorý by uchovával informáciu o tom, či sa počiatočný objekt pridaný do sveta vykonal alebo nie, a po prvom vykonaní metódy **`act`** by sa táto vlastnosť nastavila na **`true`**.

5. Algoritmus, vymenovaný typ, polia

V tejto téme sa rozoberajú ďalšie pojmy, ako sú algoritmus, vymenovaný typ (**enum**), polia a cyklus nad poľom prvkov. V tejto časti študenti implementujú karty, ktoré sa budú používať na hranie hry.

Tabuľka 16 obsahuje porovnanie podobných tém v projektoch Mravce a Bomberman. Ako je možné vidieť, projekt Mravce sa viac zameriava na tvorbu a precvičovanie a o niečo viac na akvizíciu. Aj celková pracovná záťaž je približne dvakrát väčšia ako v projekte Bomberman. Je to spôsobené tým, že táto téma pozostáva nielen z práce so zoznamami, ale zavádza aj pojmy ako vymenovaný typ atď.

Tabuľka 16: Porovnanie pracovnej záťaže algoritmus, vymenovaný typ, polia v projekte Mravce a podobnej témy v projekte Bomberman - zoznam a for each cyklus

			
Bomberman	3h 00min	Ants	6h 05min

5.1. Implementácia triedy Card

Najskôr môžete so študentmi diskutovať o tom, ako by finálna hra fungovala a navrhnúť triedu **Card**. Mali by ste prísť k riešeniu, ktoré bude obsahovať informácie o type karty, jej požiadavky, efekt a určitý popis. Potom môžete takúto triedu vytvoriť ako potomka triedy **Actor**.

5.2. Vymenované typy

V predchádzajúcej úlohe ste vytvorili typ karty ako atribút v triede **Card**. Diskutujte so študentmi o tom, aký typ by tento atribút mal mať - **String**, **int** atď. a môžete im predstaviť typ **enum**. Enum je dátový typ s konečnou množinou pomenovaných hodnôt (napr. pre dni v týždni sú to hodnoty: pondelok, utorok, streda, štvrtok, piatok sobota a nedeľa). Vytvorte s nimi vymenovaný typ **CardType** a zadajte jeho jednotlivé hodnoty.

5.3. Vetvenie – príkaz switch

Teraz by ste mali implementovať vykreslenie karty. To je založené na type karty, aby sa karty vizuálne od seba odlišovali. Môžete študentom ukázať, ako by sa to urobilo pomocou príkazu **if** a porovnať to s príkazom **switch**. Existujú tri typy kariet – stavebné karty, útočné karty a magické karty. Na základe typu karty sa vyberá jej pozadie. V prípade, že to budete robiť pomocou príkazu **if**, kód by mohol vyzerať takto:

```
if(type == BuildTower || type == BuildWall || type == IncreaseBricks) {
      background = new GreenfootImage("building-card.png");
} else if(type == IncreaseSwords || type == Attack)
```

```
...
```

Tento príkaz možno nahradiť príkazom **switch**. Príkaz **switch** v Jave funguje tak, že sa vykoná konkrétna vetva, ale nezastaví vykonávanie ostatných vetiev, pokiaľ nevyvoláte príkaz **break**. V uvedenom prípade to umožňuje zlúčiť viacero typov kariet do jednej skupiny a príkaz priradenia napísať až za posledný typ karty v skupine typov kariet. Preto tento 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
	...
}
```

možno zapísať aj spôsobom, ktorý sa používa v tejto časti:

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

5.4. Pole

Hra by mala obsahovať inštancie triedy **Card**. To možno vykonať zavedením troch polí typu **Card** (môžete tiež rozšíriť ruku – t. j. počet kariet, ktoré hra ponúkne hráčovi v jednom ťahu – na viac z nich). Môžete im vysvetliť, že by nebolo možné uložiť ešte viac kariet, keď sa rozhodne ruku ešte viac rozšíriť a môžete žiakom vysvetliť koncept polí. Mali by ste tiež vytvoriť inštanciu poľa.

5.5. Zjednodušenie vytvárania inštancií kariet - CardFactory

Pri vytváraní nových inštancii typu **Card** je potrebné uviesť množstvo informácií. Môžete sa pokúsiť nájsť riešenie tohto problému. Jedným z riešení je vytvoriť triedu **CardFactory** , ktorá bude uchovávať inštancie pre každú kartu a implementovať metódu **clone** v triede **Card**. Takto je možné vytvárať nové karty pomocou triedy **CardFactory**, konkrétne klonovaním už existujúcich kariet.

5.6. *Random – Vytvorenie inštancie náhodnej karty*

Po vytvorení triedy **`CardFactory`** a jej metódy **`clone`** by ste mali byť pripravený na ďalšiu časť. Totiž trieda **`CardFactory`** by mala vedieť vytvoriť aj náhodné inštancie triedy **`Card`**. Môžete vysvetliť generátor náhodných čísel a implementovať metódu, ktorá bude klonovať (**`clone`**) náhodnú kartu a tiež náhodnú základnú kartu (základné karty sú karty bez nákladov – to je implementované preto, aby ste mohli zaručiť, že hráč môže vždy zahrať aspoň jednu z poskytnutých kariet)

5.7. *Cyklus cez pole*

Teraz musíte v hre vytvoriť inštanciu triedy **`CardFactory`** a implementovať generovanie kariet a ich vymazanie (odstránenie zo sveta) - pri vysvetľovaní môžete zaviesť vybranú formu cyklu.

5.8. *Vykreslenie hry*

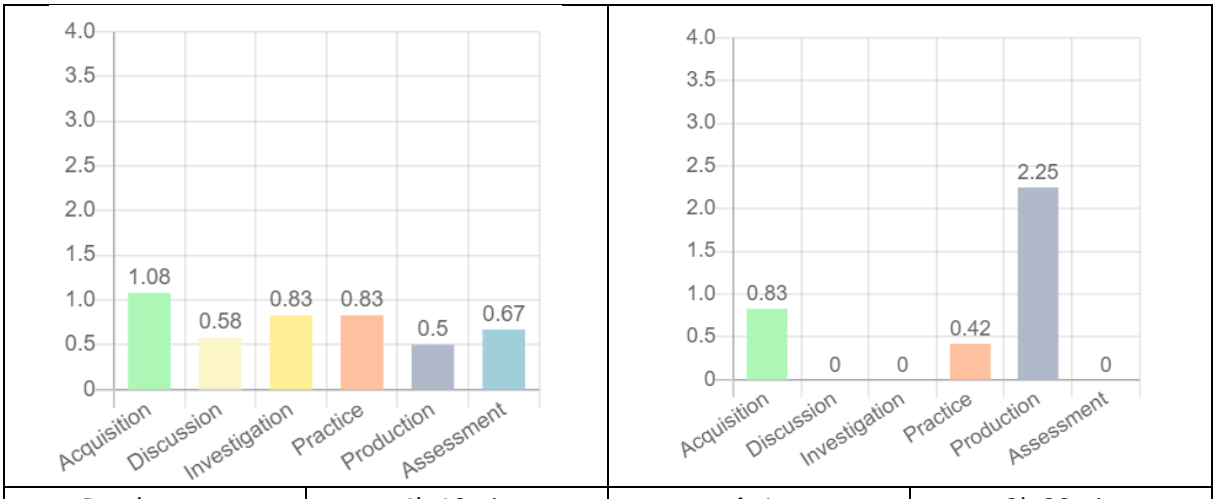
Poslednou úlohou v tejto téme je implementácia kreslenia celej hry. Počas tohto procesu by ste mali pripraviť aj karty hráčov pomocou už vytvorenej metódy a tiež zaviesť atribút pre uchovávanie informácie o tom, či je aktívny prvý hráč. Vykreslenie hry by malo pozostávať z vykreslenia všetkých kariet a vykreslenia informácie, aký hráč je práve na ťahu.

6. Spracovanie používateľského vstupu, logika hry

Táto téma sa zameriava na spracovanie vstupov od používateľa - ako od neho získať napríklad meno hráča, ako spracovať klikanie na karty a ako dokončiť logiku hry. V tejto časti sú tiež predstavené niektoré pokročilé koncepty (napr. singleton).

Tabuľka 17 ukazuje porovnanie tejto témy s najpodobnejšou témou v projekte Bomberman - algoritmus, ovládacie prvky aplikácie, vytváranie metód. Upozorňujeme, že keďže ide o poslednú tému tohto projektu, mnohé pojmy zavedené v projekte Bomberman sú v tomto bode projektu študentom už známe. Preto je tu výrazne menší dôraz na skúmanie a diskusiu ako v projekte Bomberman a oveľa viac na tvorbu.

Tabuľka 17: Porovnanie pracovnej záťaže témy spracovanie používateľského vstupu, logika hry v projekte Mravce a podobnej témy v projekte Bomberman - algoritmus, ovládacie prvky aplikácie, vytváranie metód



Bomberman	4h 10min	Ants	3h 30min

6.1. *Zadávanie mien používateľom*

V tejto úlohe by ste mali študentom vysvetliť dialógové okná (metóda **`ask`** v triede **`Greenfoot`**) a nastaviť názvy hráčov podľa týchto vstupov.

6.2. Statická inštancia triedy - Game ako singleton

Inštancie triedy **Game** by mali byť skonštruované len jedenkrát – tento problém môžete prediskutovať so študentmi a môžete poskytnúť riešenie vo forme singletonu – statickej inštancie triedy **Game** a súkromného konštruktora (**private**). Singelton (jedináčik) sa používa v prípadoch, keď chceme, aby v celej aplikácii existovala len jedna inštancia určitej triedy. Známym príkladom singletonu je napr. balík v operačnom systéme). Je to potrebné, pretože keď hráč použije kartu, mala by existovať referencia na inštanciu triedy **Game**, aby sa kliknutie na ňu dalo správne spracovať, ako sa o tom bude hovoriť v ďalšej časti. Iný spôsob implementácie (bez singletonu) je poskytnúť inštanciu triedy **Game** konštruktorom tried **Card** a **CardFactory**.

6.3. Spracovanie kliknutia na kartu

Kliknutie myšou sa spracúva volaním metódy **mouseClicked** triedy **Greenfoot** v príslušnej metóde **act()**. Po kliknutí by ste mali zavolať metódu **useCard** v inštancii triedy **Game** a poslať odkaz na seba (**this**).

6.4. Implementácia metód pre prácu s hodnotami atribútov v inštanciách tried Card a Player

Pred implementáciou zvyšnej logiky pre hru potrebujete aj ďalšie gettery a settery (nastavenie hodnoty atribútu) pre inštancie tried **Player** a **Card**. Mali by ste ich teda implementovať (pre študentov by to nemal byť problém, keďže to už bolo urobené predtým).

6.5. Implementácia podporných metód pre hráča a opravu vo svete

Keďže trieda **Game** je teraz singleton, je potrebné ju pridať do sveta v triede **MyWorld**. Ďalšou úlohou je implementovať podpornú metódu pre hráča, ktorá bude slúžiť na obdržanie určitého množstva poškodenia. Príslušný hráč by si mal podľa toho znížiť múr alebo vežu.

6.6. Implementácia hernej logiky

Nakoniec je potrebné implementovať hernú logiku reprezetovanú metódou **turn**, ktorá sa skladá z nasledujúcich krokov popísaných nižšie.

1. Skontrolujte, či jeden z hráčov vyhral – to znamená, že ak veža niektorého z hráčov dosiahne výšku 100 alebo klesne pod hodnotu 0. Ak je splnená jedna z týchto podmienok, zobrazí sa víťazná obrazovka a ukončí sa herný cyklus zavolaním príkazu **return**.
2. Prepnite aktívneho hráča na druhého – stačí znegovať hodnotu atribútu **isPlayer1Active**.
3. Pripravte karty pre hráča na ťahu – nakoľko sa už vytvorila metóda **prepareCards** v tomto kroku ju stačí len zavolať.
4. Spravujte ťah hráča – konkrétne klikanie na karty. Keďže inštancia triedy **Card** dokáže počúvať na klikanie, stačí zavolať metódu **draw**, ktorá vylosuje pripravené karty.
5. Prekreslite hráčov – vykoná sa pomocou metódy **redraw** pre každého hráča.

Potom je potrebné implementovať metódu **useCard** v tirede **Game** použitím príkazu **switch**. Ten by mal obsahovať vetvu pre každý typ karty a spracovať jej vykonanie. Existuje 7 typov kariet: **BuildTower**, **BuildWall**, **IncreaseBricks**, **IncreaseSwords**, **Attack**, **IncreaseMagic** a **StealBricks**. Ako príklad sa ukáže riešenie prvého typu karty – **BuildTower**. V tomto prípade musíte skontrolovať, či hráč môže túto kartu zahrať (t. j. počet jeho tehál je väčší alebo rovný požiadavkám karty), zvýšiť hodnotu atribútu **height** v inštancii triedy **Tower** a znížiť hodnotu atribútu **bricksNumber** podľa hodnoty na príslušnej inštancií triedy **Card**. Kód pre tento typ karty teda môže vyzerať takto:

```
if (activePlayer.getBricksNumber() >= card.getRequirements())
{
    activePlayer.increaseTowerHeight(card.getEffect());
```

```
        activePlayer.setBricksNumber(
                activePlayer.getBricksNumber() - card.getRequirements()
        );
}
```

Ostatné typy kariet sú v istom zmysle podobné:

- **`BuildWall`** – musíte skontrolovať hodnotu atribútu **`bricksNumber`** príslušného hráča a zvýšiť hodnotu atribútu **`height`** v inštancii triedy **`Wall`**,
- **`IncreaseBricks`** – nemusíte nič kontrolovať a zvýši sa hodnota atribútu **`bricksNumber`** pre daného hráča,
- **`IncreaseSwords`** – nemusíte nič kontrolovať a zvýši sa hodnota atribútu **`swordsNumber`** pre daného hráča,
- **`Attack`** – musíte skontrolovať hodnotu atribútu **`swordsNumber`** daného hráča a zavolať metódu **`receiveDamage`** neaktívneho hráča,
- **`IncreaseMagic`** – nemusíte nič kontrolovať a zvýši sa hodnota atribútu **`magicNumber`** pre daného hráča,
- **`StealBricks`** – musíte skontrolovať hodnotu atribútu **`magicNumber`** daného hráča, znížiť hodnotu atribútu **`bricksNumber`** neaktívneho hráča a navýšiť hodnotu atribútu **`bricksNumber`** aktívnemu hráčovi.

Je možné vytvoriť aj iné typy kariet - záleží na fantázii študentov. Toto sú len niektoré základné typy kariet.

Nakoniec musíte po zahraní karty zavolať metódu **`turn`**, aby sa zaistila správna logika hry.

Ako posledný bod hry by ste mali implementovať víťaznú obrazovku. Tá je rovnako ako ostatné obrázky v tomto projekte na vás, takže buď si ju vytvoríte spolu so študentmi, alebo ju necháte na nich.

Ešte je potrebné vykonať drobné úpravy kódu v triedach **`Player`** a **`Tower`** kvôli čistote kódu.

4. Zoznam použitej literatúry

[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. Prílohy

5.1. Export návrhu vzdelávania pre projekt Bomberman

Pozrite si súbor LD_Bomberman.pdf

5.2. Export návrhu vzdelávania pre projekt Tower defense

Pozrite si súbor LD_Tower_defense.pdf

5.3. Export návrhu vzdelávania pre projekt Mravce

Pozrite si súbor LD_Ants.pdf