Lehrkonzeption für die Programmierausbildung nach LIGHT-OOP-Prinzipien

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

Projekt	Object-oriented Programming for Fun (Objektorientierte Programmierung mit Spaß)
Kurzbezeichnung	OOP4FUN
Vertragsnummer	2021-1-SK01-KA220-SCH-00027903
Projekt-Koordinator	Žilinska univerzita v Žiline (Slowakei)
Projekt-Partner	Sveučilište u Zagrebu (Kroatien) Srednja škola Ivanec (Kroatien) Univerzita Pardubice (Tschechische Republik) Gymnazium Pardubice (Tschechische Republik) Obchodna akademia Povazska Bystrica (Slowakei) Hochschule für Technik und Wirtschaft Dresden (Deutschland) Gymnasium Dresden-Plauen (Deutschland) Univerzitet u Beogradu (Serbien) Gimnazija Ivanjica (Serbien)
Jahr der Herausgabe	2023

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

1. Überblick

1.1. Beschreibung des Arbeitsgebiets

Es geht darum, Programmieraufgaben mit Mitteln der objektorientierten Programmierung (OOP) zu lösen. Dabei wird dem Light-OOP-Paradigma gefolgt, das nur Kernkonzepte der OOP beinhaltet, die oft intuitiv zugänglich sind. Die Schüler lernen, gegebene Aufgaben zwischen kooperierenden Objekten aufzuteilen, ihre Kompetenzen zu bestimmen und ein entworfenes Modell umzusetzen. Der Kurs erfordert keine Vorkenntnisse bezüglich Programmierung. Es wird in der Programmiersprache Java gelehrt. Der Kurs erklärt die Light-OOP-Konzepte (wie Kapselung, Vererbung oder Assoziation) anhand der Erstellung von einfachen Computerspielen, bei denen diese Konzepte einfach und intuitiv genutzt werden. Der Prozess der Erstellung eines Computerspiels basiert auf Teamarbeit und nutzt praktisches Wissen und Fähigkeiten aus anderen Bereichen der Informatik und der damit verbundenen Themen (auch verbunden mit der Arbeit mit Multimedia- und Bürosoftware). Das Design jedes Computerspiels ist offen genug, um das Spiel individuell und kreativ zu erweitern. Darüber hinaus führt das Design zur fachgerechten Nutzung erworbenen Wissens.

1.2. Besonderheiten des Arbeitsgebiets

Das Subjekt konzentriert sich darauf, einen innovativen Ansatz zur Lehrprogrammierung einzuführen, der auf der Lösung von Aufgaben mit dem objektorientierten Programmierparadigma (OOP) basiert. OOP ist heute das dominierende Paradigma für die Anwendungsentwicklung. Daher ist es für Studenten angemessen, das Wissen und die Fähigkeiten in diesem Bereich zu besitzen. Das Thema stellt eine Entwicklungsumgebung dar, die verschiedene Formen der Quellcode-Bearbeitung (Frame-basiertes Editieren mit vereinfachter Form sowie echtes Quellcode-Schreiben) verwendet, was es ermöglicht, Studenten auf verschiedenen Ebenen des technischen Wissens und der Aktivität zu unterrichten. Mit seiner Einfachheit und Klarheit unterstützt dieses Tool ein schnelles und intuitives Verständnis von gelehrten Themen, was positiv auf die Schüler und ihre Motivation beeinflusst.

1.3. Zweck

Durch die Programmierung interaktiver Spiele in einer grafischen Umgebung entwickelt der Schüler Kenntnisse und Fähigkeiten, so dass der Schüler in der Lage sein wird:

- Problem zu identifizieren,
- Subjekte zur Lösung des identifizierten Problems (Objekte) zu bestimmen,
- Designklassen von Objekten sowie deren Attribute und Methoden festzulegen,
- die Beziehungen der Objekte (Assoziierung, Vererbung) zu identifizieren und richtig zu nutzen,
- einen Algorithmus zu entwerfen, um das Problem zu lösen und es unter kooperativen Objekten zu verteilen,
- Quellcode-Elemente (Verzweigung, Schleifen) zu verwenden, um den entwickelten Algorithmus zu implementieren,
- Techniken für Code-Debugging zu verwenden,
- eine einfache Anwendung mit grafischer Schnittstelle in der Greenfoot-Umgebung zu erstellen.

1.4. Lernziele

Die Lernziele werden wie folgt zusammengefasst:

- das Verständnis der Grundprinzipien der objektorientierten Programmierung,
- das Verständnis der Grundlagen von Algorithmen,
- das Verständnis der Syntax der Java-Programmiersprache,
- Fähigkeit der Analyse einer Programmausführung basierend auf dem Quellcode,

- die Fähigkeit, eigene Programme mit dem Einsatz von OOP zu erstellen.

1.5. Geräte und technische Anforderungen

Vorausgesetzt wird ein PC-Kabinett mit einem separaten PC und Arbeitsbereich für jeden Schüler, sowie ein Arbeitsraum für Lehrer. Alle PCs sollten mit einem LAN-Netz verbunden sein. Ein Internetzugang wird empfohlen. Die PCs sollten die folgenden Mindestanforderungen erfüllen:

- Betriebssystem in einer aktuellen Version (Microsoft Windows, Linux, Mac OS),
- Bürosoftware mit Textverabeitung, Tabellenkalkulation und Präsentationssoftware (z.B. Microsoft Office, Libre Office, Open Office),
- Java SE Entwicklungskit (JDK),
- Greenfoot Umgebung (Version 3.8 oder höher),
- einfache Software zur Grafikbarbeitung, bspw. zur Erstellung von PNG-Bildern
- Webbrowser (z.B. Edge, Google Chrome, Mozilla Firefox, Opera).

2. Prinzipien der Lehrkonzeption

Der vorgeschlagene Lehrplan soll die Probleme in PR1 und PR2 berücksichtigen (siehe Kapitel "Ergebnisse mit PR1-Ergebnis bestehen" im PR2-Bericht). In der folgenden Tabelle präsentieren wir die Perspektive auf die Entwicklung des Lehrplans, die Lernergebnisse, die Lehrmaterialien und die Lehrtätigkeit, so wie in PR2 vorgeschlagen. Nach diesen Erkenntnissen konnten wir die Lehrplanprinzipien formulieren.

PR2 Erkenntnisse	PR3 Prinzipien der Lehrkonzeption
In den High Schools sollte OOP mit allgemeinen Programmieraufgaben kombiniert werden, welche die grundlegenden Programmierkonzepte am Anfang abdecken. Speziellere Themen im Zusammenhang mit OOP wären in separaten Kursen angemessener. Es ist von entscheidender Bedeutung, den Informationsaustausch zwischen Schul- und Universitätslehrern zu verbinden und zu fördern, unter Beteiligung von politischen Entscheidungsträgern, die Lehrpläne im Zusammenhang mit Programmierkenntnissen auf allen Bildungsstufen definieren. Aus der Perspektive des Kurses, des Lehrers und aus der Kursdesign-Perspektive sollten folgende innovative Formen des Unterrichts/Wissenstransfers verwendet werden: Blended Learning, Learning-by-Doing, Problemlösung, kollaboratives Problem, Teamwork, problembasiertes Lernen, aktives Lernen, laborbasiertes Lernen. Darüber hinaus sollten verschiedene Formen innovativer Ansätze in Vorträgen, Seminaren und Laborübungen angewendet werden. Spiele und Gamification im Allgemeinen wurden häufig verwendet, um Studenten zum Programmieren zu motivieren. Schülern gefällt es, kreativ zu sein oder um Wissen zu konkurrieren, wenn sie durch ein angemessenes Setting unterstützt werden. Aus den Ergebnissen der Literaturrecherche wurden drei verschiedene Arten des Lernens durch Spiele vorgeschlagen: Lernen durch Spielen, Lernen durch die Erstellung von Spielen, Lernen durch die Verwendung von spielbezogenen Tools und das Lernen mit Gamification.	Y Syllabus muss OOP von Anfang an nach dem ersten Objektansatz richtig nutzen. Das geeignete OOP-Niveau für Gymnasien wurde als leichtes OOP identifiziert und formuliert. Light OOP kann mit Vorteilen bei der Erstellung von Spielen verwendet werden, da es einfach ist, die Objekte des Spiels sowie die Kompetenzen und Eigenschaften der Objekte zu identifizieren. Um Studenten mit Spielen zu motivieren, ist es wichtig, die Schüler für die Spiele zu interessieren. Es sollte mehrere spielbasierte Projekte mit verschiedenen Mechaniken entstehen, um dieses Ziel zu erreichen. Darüber hinaus werden mehrere Spiele, die vorbereitet werden, zulassen: - Erstellen Sie verschiedene Unterrichtsmaterialien, die auf verschiedene Unterrichtsbedingungen abzielen (online/onsite, intensiver/ganzjähriger Kurs, Anfänger/fortgeschrittene Studenten). Dies ist eine wichtige Eigenschaft für die anstehenden Projektergebnisse, - Bauen Sie ein Spiel mit dem anwesenden Lehrer und anderen Spielen als Heimaufgaben (Lehrer kann sehen, ob die Schüler Wissen in verschiedenen Kontexten anwenden können), - Erstellen Sie ein Spiel gemäß den gegebenen Anweisungen mit minimaler Lehrerinteraktion, so dass die Schüler die Bedeutung der gut geschriebenen technischen Beschreibung verstehen und sie werden gelernte Fähigkeiten anwenden, um Spiel zu schaffen, wie angegeben, - Einführung eines neuen leichten OOP-Konzepts, da das Spiel neue Mechaniken

Für das Lernen und Lehren von OOP-Konzepten zeigte das Lernen durch die Erstellung der Spiele die signifikanten Auswirkungen, um die Fähigkeiten der Schüler zu verbessern und sie in eine unterhaltsame und unterhaltsame Umgebung einzubinden.	einführt. Dies wird es ermöglichen, die Entwicklung des Projekts zu stoppen, wenn der Lehrer beschließt, die Teilmenge des leichten OOP wegen der Besonderheiten der Nation abzudecken.
Wie mehrere PR2 Ergebnisse zeigen, sollte das Hauptziel darin liegen, Lern- und Lehraufgaben in anregende und unterhaltsame Aktivitäten zu integrieren, die sich positiv auf höhere Teilnahme- und Abschlussquoten auswirken werden. Dies würde das Interesse der High-School-Schüler für die Programmierung im Allgemeinen erhöhen und schließlich zu einem besseren Verständnis von Programmierung und OOP-Konzepten führen. In diesem Fall wären die Studenten nicht mehr überfordert, wenn sie mit den Lehrplänen der Universität konfrontiert werden.	Projekte werden nach dem Learning-By-Doing-Prinzip gestaltet. Wir versuchen, die Theorieerklärung zu minimieren und untersuchen, Praxis und Produktionsaspekte des Lernens zu unterstützen. Die Arbeit in Gruppen ermutigt Studenten, die am Anfang Schwierigkeiten haben können. Wenn das Projekt in einer Gruppe entwickelt wird, können agile Methoden der Entwicklung sowie des Unterrichts genutzt werden. Vor der Umsetzung kann der Lehrer die Analyse- und Designphase des Projekts nutzen. Dies wird es ermöglichen, Kenntnisse und Fähigkeiten von Studenten zu nutzen, die in früheren Vorlesungen/Kursen gewonnen wurden, sowie die Studenten in die Projektentwicklung zu bringen. Die Schüler können gebeten werden: - mit Informationsquellen z uarbeiten, um das richtige Spiel zu finden, - Spielregeln und/oder Anforderungen an ihre Anwendung zu formulieren (entweder in mündlicher oder schriftlicher Form), - multimediale Inhalte (Bilder/Klang) vorzubereiten.
Das übergeordnete Ziel von PR2 war es, geeignete und innovative Lern- und Lehrideen und -ansätze zu finden, die diese Probleme lösen würden. Wie in den vorherigen Kapiteln erwähnt, gibt es mehrere bekante gute Praktiken, die verwendet werden könnten, um die Erreichung der Lernergebnisse während der High-School-Ausbildung zu verbessern. Es sollte jedoch auch darauf hingewiesen werden, dass die Neugestaltung des Lehrplans zur Einführung von OOP-Themen führen und Ziele bei der Erreichung der OOP-bezogenen Lernergebnisse setzen sollte. Es ist auch wichtig, eine angemessene Art von Bewertung auszuwählen: (Online-)Fragebögen sind die einzige akzeptierte Methode zur Beurteilung des Gefallens, Nutzen, Interesse, Engagement und Vereinfachung der	Nr. Y3 Mit dem Fokus auf Learning-By-Doing haben wir versucht, die Art des Erwerbs von Aktivitäten zu minimieren und den Untersuchungsteil zu stärken. Projekte werden so gebaut, dass eine einfache Erweiterung ermöglicht, so dass motivierte Studenten in der Lage sein werden, selbst an den Projekten zu arbeiten. Um den vorgeschlagenen Lehrplan zu validieren, bereiten wir das Lerndesign für jedes Projekt vor. Dann vergleichen wir die Analyse der Analyse von neuen Projekten mit Licht OOP mit analytischem Ergebnis des für in der Vergangenheit entwickelten Projektdesigns, das bereits in der Praxis in der Slowakei eingesetzt wird (u.a. in der Slowakei (u.a. Schulen im Land wird es von Projektpartner Obchodna akademia

<table>
<tr><td>Programmierung und OOP-Konzepte, die auf der High-School- und nicht auf Universitätsebene definiert werden.</td><td>Povazska Bystrica) und Tschechien (verwendet vom Projektpartner Gymnazium Parubdica) verwendet. Positives Feedback für dieses validierte Projekt wurde in der PR2 veröffentlicht, daher gehen wir davon aus, dass wir nach den gleichen besten Prinzipien und Praktiken die positive Akzeptanz des vorgeschlagenen Lehrplans übertragen werden.</td></tr>
<tr><td>Durch den Einsatz von Teamwork in OOP-Aufgaben hätten die Schüler die Möglichkeit, ihr Wissen zu teilen und die Umsetzung grundlegender Programmierkonzepte auf andere Studenten (Peer-to-Peer-Learning) zu übertragen.</td><td rowspan="2">Bei der Entwicklung hier vorgeschlagener spielbasierter Projekte haben wir die Nutzung verschiedener Techniken wie EduScrum im Auge behalten. Aus diesem Grund liefern wir jedes Projekt in Form von GIT-Repositorium und liefern den Lehrern das richtige Wissen. Während es nicht zwingend ist, diesen Ansatz zu nutzen und Lehrer noch traditionellen Ansatz verwenden können, ist es dennoch vorteilhaft um:

* Grundlagen von Versionierungssystemen einzuführen - wir schlagen GIT als am meisten genutztes Versioniersystem vor, das es einfach macht, es zu verwenden
* Quellcodes zu verteilen und arbeitsteilig zu bearbeiten und
* neue Features der Spiele entwickeln, ohne Einfluss auf den Projektfluss (z.B. in einem separaten Zweig), was ambitionierte Studenten motivieren wird.
* Teamwork – jeder Student wird für den spezifischen Aspekt eines Spiels verantwortlich sein, das zur Notwendigkeit einer effektiven Kommunikation zwischen den Teammitgliedern führt.
* Zeitmanagement – einzelne Teile der Projekte müssen pünktlich geliefert werden, um sie zu verschmelzen und mit anderen Arbeiten fortzufahren, aber die richtige Nutzung des Versionierungssystems und OOP eröffnen viele Möglichkeiten, mit Zeitkämpfen umzugehen, was auch in schwierigen Zeiten zu einer positiven Motivation der Studenten führen kann.

Wir werden Multiplikatoren-Events mit Lehrern realisieren, die sowohl die Einführung in den GIT als auch die Nutzung von Prinzipien der agilen Softwareentwicklung im Unterricht abdecken. Die Studenten werden Projektteams bilden, mit</td></tr>
<tr><td>Die Ergebnisse dieses Projekts werden eine Reihe von Materialien liefern, die hoch motivierten Studenten mit fundiertem Vorwissen die Chance geben, dieses Wissen durch verschiedene Aktivitäten und Rollen zu verbessern. Solche Studenten könnten die Gesamtleistungen der gesamten Gruppe erheblich verbessern, wenn sie die Möglichkeit erhalten, das Wissen zu teilen oder die Teams zu führen.</td></tr>
</table>

	bestimmten Rollen, sie werden Ideen zu Stand-Ups teilen, sich Ziele geben und ihnen zuweisen, sie werden Sprints realisieren, Dokumentationen und andere Artefakte erstellen und sie werden ihre Lösung präsentieren.
Der Einsatz verschiedenerer Tools und Programmiersprachen in früheren Studienjahren sollte gefördert werden. Programmierkonzepte könnten mit Visualisierungstools vereinfacht werden. Obwohl einige Länder bereits die Verwendung einiger Tools wie Logo oder Scratch eingeführt haben, sind diese für Grundschulen interessant, aber nicht für High Schools. Daher sollten fortschrittlichere Werkzeuge verwendet werden, die zur Unterstützung von OOP entwickelt wurden. Wir haben erkannt, dass Alice und Greenfoot in diesem Aspekt führend sind.	Wir benutzen die Greenfoot-Umgebung, die auf der Java-Programmiersprache aufbaut. Java ist derzeit sehr beliebt und eine in der Praxis sehr weit verbreitete Programmiersprache. Darüber hinaus stellt Greenfoot den framebasierten Quellcode-Editor mit der Sprache Stride bereit. Das erlaubt Lehrern, die in diesem Lehrplan vorgestellten Techniken auch bei Schülern jüngeren Alters einzusetzen. Greenfoot ist sehr visuell und von Anfang an ermöglicht es, ein visualisiertes Objekt zu schaffen, das "alive" ist und mit ihm interagiert werden kann. Daher wird die theoretische Einführung minimiert, und die Schüler können von Anfang an mitarbeiten.
Wie die Schüler berichteten, ist es für den Unterricht wichtig, dass Lehrer neue und aktuelle Unterrichtsmaterialien verwenden und kreative Lehrmethoden anwenden. Auch die Verfügbarkeit und Flexibilität des Lehrers it nötig, mit Schülern außerhalb des Klassenzimmers zu arbeiten, um die Schüler zu motivieren und ein größeres Interesse an dem Fach zu wecken.	Mit den vorgestellten Prinzipien schaffen wir einen modernen Lehrplan für Lehrer, die leichte OOP-Themen abdecken und auf Projektarbeit basieren und wirklich Objekt-First-Ansatz nutzen. Wir schlagen mehrere spielbasierte Projekte vor, die jeweils in Form von GIT-Repository geliefert werden. Für jedes Projekt entsteht Lerndesign, das es ermöglicht, diese Projekte zu validieren, wobei bereits in der Praxis genutzt und positiv aufgenommener Ansatz zu validieren.

Inhalt und Umfang des Bildungsprogramms unterscheiden sich in Bezug auf das Spielprojekt, das während des Unterrichts entwickelt wurde. Für detaillierte Analysen wird auf das jeweilige Lerndesign und die angehängten Analysedateien verwiesen.

3. Programmierprojekte

Es wurden drei Spielprojekte geschaffen, die den Prinzipien des objektorientierterten Denkens und Learning-By-Doing folgen. Darüber hinaus haben wir das Projekt Bomberman verarbeitet, das war ein Rückgratprojekt der nationalen Projekt-IT-Akademie, das in der Praxis in der Slowakei verwendet wird. Wir nutzen dieses Projekt zur Validierung der geplanten Projekte. Organisation aller Projektkapitel ist wie folgt.

- Projektbeschreibung - die grundlegende Beschreibung des Spiels mit Screenshot der fertigen Anwendung und zusammengefassten Spielregeln. Die Projektbeschreibung stellt auch die Verbindung zu den Licht-OOP-Themen vor.
- Link zu Quellcodes – Quellcodes werden in Form von GIT [1] Repository organisiert. Die Verwendung des ordnungsgemäß verwalteten Repositorys führt den Lehrer ein, um den modernen Ansatz des Quellcode-Managements zu verwenden. Wir haben GIT als Versionierungssystem verwendet, das zu den beliebtesten in den letzten Jahren gehört [2]. GIT ermöglicht es uns auch, cloudbasierte Repositorys wie GitHub [3] oder GitLab [4] zu verwenden, die kostenlos sind und viele Tools verwenden, um die Teamzusammenarbeit zu verbessern. Jedes Repository ist wie folgt organisiert:
    - Das Thema "Branch" - Aufgaben jedes Themas aus dem Lehrplan werden in einem eigenen Zweig entwickelt. Master branch enthält nur Initialisierungs- und Fusionen von Themenzweigen.
    - Commit pro Aufgabe - jede Aufgabe, die auf die Herstellung/Änderung des Quellcodes ausgerichtet ist, ist in Form von Commit. Beschreibung der Commit-Berufsnummer.
- Link zum Lerndesign – Lehrplan ist in Form von Lerndesign gebaut. Learning Design ermöglicht es uns, Lernergebnisse zu definieren (die notwendige Kompetenzen abdecken, die in PR1 und PR2 identifiziert wurden) und sie mit Themen zu verknüpfen (siehe Verbindung zu den Zweigen des jeweiligen GIT-Repositoriums). Die Themen sind in Einheiten organisiert, die sich aus den TLAs zusammensetzen (siehe Verbindung zu den Verpflichtungen im jeweiligen GIT-Repository). Die Verwendung von Lerndesign ermöglicht es, die Zeitzuordnung zu analysieren, was direkt mit der Validierung des erklärten Lern-für-doing-Prinzips verbunden ist. Da das Lerndesign auch für bereits etablierte und pädagogische Praxis-Nutzungsprojekt (Bomberman) verarbeitet wurde, ermöglicht es, die möglichen Probleme im Design zu identifizieren. Um Validierung durchführen zu können, werden die Projekte auf die gleiche Weise zu Themen geordnet.
- Abgedeckte Themen von Light-OOP.
- Inhalt und Umfang des Bildungsprogramms – Überblick über die Lernleistungslast in spezifischer Lernart sowie Überblick über den Beitrag der Themen zu individuellen Lernergebnissen.
- Liste der Themen. Jedes Thema enthält:
    - Kurzbeschreibung,
    - Vergleich von Lerndesigns,
    - Liste der Aufgaben.

3.1. Projekt Bomberman

Bomberman ist ein ziemlich bekanntes Multiplayer-Spiel. Das Spiel findet in einer Arena statt, die sowohl die Spieler als auch einige feste Hindernisse enthält. Der Spieler kann Bomben pflanzen, die nach einer gewissen Zeit explodieren. Ziel des Spiels ist es, die Gegner durch Bomben zu eliminieren. Einige der Hindernisse in der Arena können mit Bomben zerstört werden. Nach der Zerstörung eines Hindernisses kann ein zufälliger Bonus im Spiel erscheinen, der zum Beispiel die Geschwindigkeit oder Kraft der Bomben des Spielers erhöht. Das Spiel endet, wenn nur noch ein Spieler im Spiel ist, in diesem Fall gewinnt dieser Spieler, oder wenn es keine Spieler mehr im Spiel gibt, in diesem Fall endet das Spiel unentschieden.

Das Bomberman-Projekt behandelt die meisten Themen von Light-OOP. Es konzentriert sich auf die Hauptaspekte der OOP, die in den folgenden Themen zusammengefasst sind. Darüber hinaus führt es einige Themen ein, die schon über Light-OOP hinausgehen, wie das Generieren und die Verwendung von Zufallswerten mit der Random-Klasse.

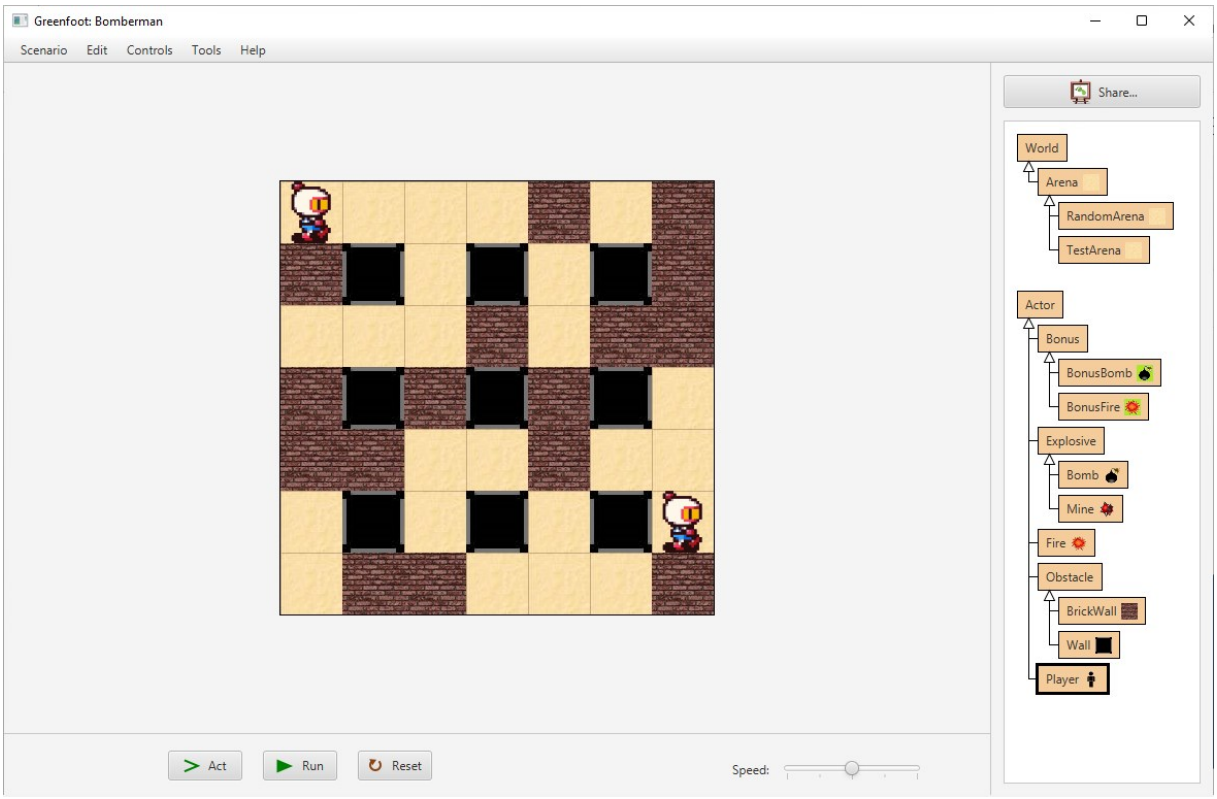


Abbildung 1: Greenfoot-Umgebung mit dem vollständigen Ausbaustand des Projekts Bomberman

Die Quellcodes sind verfügbar unter:

https://gitlab.kicon.fri.uniza.sk/oop4fun/project-bomberman

Das Lerndesign ist unter der folgenden URL verfügbar:

https://learning-design.eu/de/preview/70bcf65d805b0603f6c1aeab/Details

3.1.1. Inhalt und Umfang des Projekts

Der gesamte Zeitaufwand wird mit 49h 30min beziffert. Die Arbeitszeit ist wie folgt verteilt:

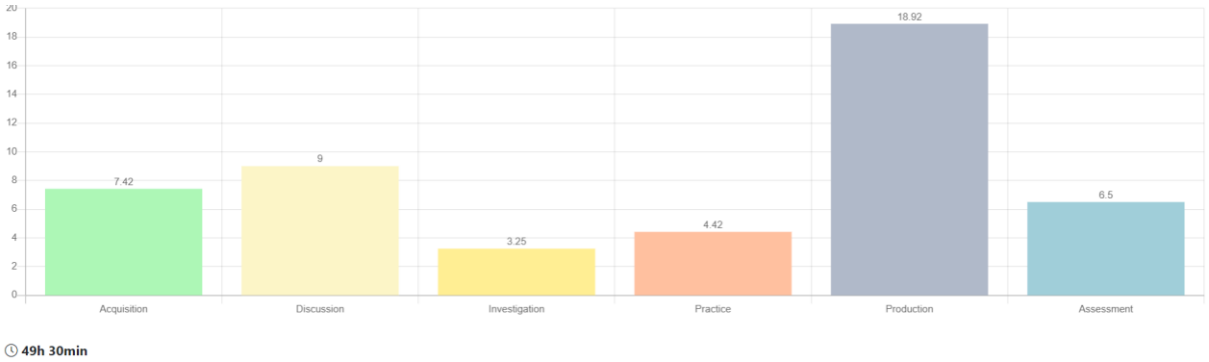


Abbildung 2: Arbeitsbelastung der Lernenden bei der Nutzung des Projekts Bomberman

Die konstruktive Ausrichtung ist in der folgenden Tabelle zusammengefasst:

Tabelle 1: Zuordnung der Themen zu unterschiedlichen Kompetenzgebieten

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						

Für mehr Details verweisen wir auf den Anhang, Gliederungspunkt 5.1.

3.1.2. Lerneinheiten / Themen

Das Projekt Bomberman gliedert sich in zehn Themen:

1. Einführung in die Greenfoot-Umgebung
2. Algorithmen, Anwendungskontrollen, Methodenerstellung
3. Verzweigung und Spieler
4. Variablen, Ausdrücke und erweiterte Spielersteuerung
5. Objekt- und Klassenkooperation
6. Vererbung und Schleife
7. Liste und For-Each-Schleife

8. Private Methoden und While-Schleife

9. Polymorphismus

10. Zufallszahlen

Die aus dem Bereich Light-OOP behandelten Themen sind:

- Klassen, Objekte, Instanz

- Methoden, Argumente für die Parametrierung von Methoden

- Konstruktor

-Attribut

- Datenkapselung

- Vererbung

- abstrakte Klassen

- Schnittstelle

- Lebenszyklus der Objekte

AB HIER V2 vom 15.8. eingebaut ----------

1. Einführung in die Greenfoot-Umgebung

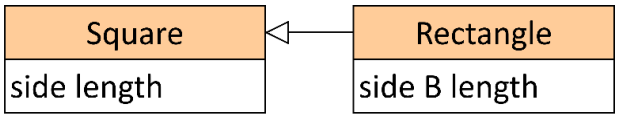
Das Thema widmet sich dem grundlegenden Projekt-Setup. Die Schüler lernen, wie man die Abmessungen und das Aussehen der Umgebung setzt, eine Klasse (als Unterklasse der Actorklasse) schafft, ihre Instanz erstellt, Methoden aufruft und ihren inneren Zustand beobachtet.

1.1. Identifikation von Objekten

In diesem Schritt sollen Objekte in Ihrer Umgebung identifiziert werden und ihre Eigenschaften und Aktionen aufgelistet werden, die sie ausführen können. Können Sie Objekte identifizieren, die keine Eigenschaften haben? Können Sie Objekte identifizieren, die nichts können? Können Sie immaterielle Objekte identifizieren (solche, die wir physisch nicht berühren können)?

1.2. Bewertung einer Klassenhierarchie

Die folgende Abbildung zeigt die Hierarchie der Klassen Square (Quadrat) und Recangle (Rechteck). Ist das eine gute Hierarchie?



1.3. Erstellen einfacher KLassenhierarchien

Erstellen Sie zur Übung jeweils Klassenhierarchien von:

a) Transportmitteln,

b) Tieren

c) und Computerteilen.

Listen Sie die Eigenschaften jeder Klasse auf. Welche Klassen sind im Design nicht materiell, d.h. man könnte keine Objekte bilden, die man anfassen kann? Wir werden solche Klassen in Zukunft als abstrakte Klassen bezeichnen.

1.4. Erstellen des Hintergrundbilds des Welt-Objekts

Es geht hier um das Erstellen einer s.g. Fliese (vgl. kleines Bild) in einem grafischen Editor, die grafisch eine Zelle der Welt (Instanz der MyWorld-Klasse) darstellt. Wählen Sie quadratische Darstellung, idealerweise 60x60 Pixel. Importieren oder speichern Sie das Bild im Projektordner im Bilderordner. Stellen Sie das Bild als Hintergrundbild der World-KLasse ein. Dies geschieht durch ein Rechtsclick auf MyWorld und das Auswählen von "Set Image". Überprüfen Sie danach, dass die MyWorld-Klasse ein kleines Miniaturbild im Klassendiagramm erhalten hat, um ihre grafische Form darzustellen.

1.5. Erzeugen der Greenfoot-Welt

Ändern Sie den MyWorld-Klassenkonstruktor, um eine 25x15-Zellwelt zu schaffen, wobei jede Zelle 60x60 Pixel groß ist. Wie müsste das Bild geändert werden, damit die Welt wie ein Schachbrett aussieht (d.h. mit abwechselnden verschiedenfarbigen Quadraten)?

Als nächstes erstellen Sie eine Spielerklasse. Dies geschieht durch Rechtsklick auf die Actor-klasse und die Auswahl der neuen Unterklasse. Geben Sie der Unterklasse einen sinnvollen Namen und wählen Sie sein Bild aus der Bibliothek. Danach ist die neue Klasse mit einem kleinen Icon-Bild ausgestattet. Um eine Instanz der Spielerklasse zu erstellen, klicken Sie mit der rechten Maustaste auf die neue Spielerklasse, wählen Sie neuen Spieler und verschieben Sie das Symbol mit dem Spieler in den Hintergrund der Welt. Mit einem Linksklick im World-Fenster wird die Instanz platziert. Um den Status einer Instanz anzuzeigen, klicken Sie mit der rechten Maustaste darauf und wählen Sie Inspect.

1.6. Inspektion des Zustands von Spieler-Objekten

Wählen Sie sich die erstellte Instanz der Spielerklasse mit der Maus aus und verschieben Sie sie auf eine andere Position in der Welt. Schauen Sie sich mittels Live-View den inneren Zustand des Objekts an– was sehen Sie? Erstellen Sie eine weitere Instanz der Spielerklasse und sehen Sie sich auch ihren inneren Zustand an. Ziehen Sie dann eine der beiden Instanzen mit der Maus auf eine andere Position – welcher Innenzustand hat sich geändert?

1.7. Interaktion mit Spielerobjekten

Rufen Sie Methoden auf, die in der Greenfoot-Umgebung über verschiedene Instanzen der Spielerklasse ausgewählt werden können. Klicken Sie dazu mit der rechten Maustaste auf die Instanz und wählen Sie z.B. die Methode move(int). Wenn Sie dazu aufgefordert werden, geben Sie eine ganze Zahl ein. Beobachten Sie, wie sich der interne Zustand der Instanz ändert.

2. Algorithmus, Steuerung der Greenfoot-Umgebung, Erstellen von Methoden

Dieser Abschnitt behandelt das Erzeugen öffentlicher Methoden, die den Spieler in der Welt bewegen. Es führt auch in Greenfoot-Umgebungstools ein, die die Ausführung der Anwendung steuern.

2.1. Angabe von einfachen Algorithmen

Notieren Sie zu Übung das Abläufe, wie man Kaffee zubereitet, wie man zur Schule geht und wie man das Mittagessen kocht.

2.2. Angabe eines allgemeingültigen Algorithmus

Schreiben Sie einen allgemeinen Algorithmus für die Zubereitung eines heißen Getränks. Denken Sie darüber nach, was die Eingaben eines solchen Algorithmus sein müssen, damit es allgemein ist.

2.3. Ansicht und Erweiterung einer Klasse

Erkunden Sie die Methoden der Spielerklasse. Klicken Sie dazu mit der rechten Maustaste auf die Klasse und wählen Sie Open Editor. Was beobachten Sie? In Analogie zur act()-Methode fügen Sie die makeLongStep()-Methode hinzu.

2.4. Angabe einer neuen Methode

Fügen Sie eine Anweisung zum Körper der makeLongStep()-Methode hinzu, so dass die Instanz der Spielerklasse entlang zwei Zellen in der aktuellen Richtung bewegt wird. Dann erstellen Sie mehrere Instanzen der Spielerklasse und rufen Sie diese Methode in jeder Instanz an. Beobachten Sie, ob sich die Objekte wie erwartet verhalten.

2.5. Erstellen einer Dokumentation für eine Methode

Fügen Sie einen Dokumenations-Kommetar für die `makeLongStep()`-Methode ein.

2.6. Erstellen einer Dokumentation für eine Klasse

Bearbeiten Sie den Dokumentationskommentar der Spielerklasse. Fügen Sie die Version der Klasse und ihren Autor hinzu.

2.7. Ansehen der Dokumentation

Öffnen Sie das Documentation-Fenster und schaunen SIe sich an, wie Ihre in den vorherigen Schritten eingegebenen Zeilen zur Dokumentation jetzt angezeigt werden.

2.8. Hinzifügen einer Methode für die Spieler-Klasse

Ändern Sie den Quellcode der act()-Methode in der Player-Klasse, damit darin die makeLongStep()-Methode aufgerufen wird.

2.9. Benutzen der Greenfoot-Anwendungssteuerung

Probieren Sie die Tasten aus, die die Greenfoot-Anwendung steuern. Erstellen Sie mehrere Instanzen der Spielerklasse. Drücken Sie den Act-Button – was passiert? Drücken Sie die Run-Taste – was passiert? Was passiert nach dem ersten Druck der Run-Taste, was passiert dann? Welche Auswirkungen hat der Speed-Schieber auf den Act()-Methodenaufruf nach dem Ausführen-Schaltnopf? Was passiert, wenn Sie die Reset-Taste drücken?

2.10. Programmierung einer weiteren Aktion des Spielers

Fügen Sie der Spielerklasse eine Methode hinzu, die eine Instanz der Spielerklasse einen quadratischen Weg ablaufen lässt. Dabei sind 5 Zellen vorwärt zu gehen, eine 90 Grad-Drehung zu veranlassen, dann wieder 5 Zellen vorwärts zu gehen, zu drehen, u.s.w.. Dokumentieren Sie Ihre Methode. Verwenden Sie geeignete Methoden aus der Actor-Basisklasse, um die Objekte zu bewegen und zu drehen. Ändern Sie die act()-Methode so, dass eine Instanz der Actor-Klasse sich entlang der oben beschriebenen quadratischen Wegform weiter bewegt, wenn sie aufgerufen wird. Dann verifizieren Sie Ihre Lösung, indem Sie Ihre Anwendung ausführen.

3. Programmverzeigungen und Steuerung der Spieler-Objekte

Dieses Thema führt Studenten in Benutzung von Fallunterscheidungen mit if-else Statements und der switch-Mehrfachauswahl ein. Beide Möglichkeiten werden für Programmverzeigungen benutzt. In der Anwednung werden Verzweigungen bei der Erkennung der Ränder der Welt und bei Kollisionen mit Wänden verwendet.

Der folgende Zwischenschritt umfasst eine Änderung der act()-Methode, um eine Instanz der Spielerklasse ein Feld weiter zu bewegen und zu drehen, wenn sie den Weltrand berührt.

3.1. Bewegen des Spielers über die Tastatur

Ändern Sie den Code der act()-Methode in der Spielerklasse so, dass sich der Player nur bewegt, wenn die M-Taste gedrückt wird (M als Schritt). Berücksichtigen Sie im Code, dass der Spieler gedreht wird, wenn er den Rand der Welt erreicht. Denken Sie über seinen Standort nach. Wann kann ein Spieler abbiegen?

3.2. Erkundung des Zustands eines Spieler-Objekts

Erstellen Sie eine Instanz der Spielerklasse und legen Sie sie in die Mitte des Boards. Öffnen Sie ein Fenster mit dem internen Zustand der Instanz und positionieren Sie es so, dass es während des laufenden Einsatzes sichtbar ist. Dann führen Sie die Anwendung aus und beobachten Sie, wie sich die Werte der x- und y-Attribute im Spielerobjekt ändern. Wie verändern sich diese Werte, wenn Sie sich nach oben, unten, links und rechts bewegen? Verwenden Sie verschiedene Werte für die setRotation()-Methode (0, 90, 180, 270). Versuchen Sie, die Methode isAtEdge() durch eine geeignete Benutzung der getX() und getY()-Methoden zu ersetzen.

3.3. Erkennung der Ränder des Weltbereichs

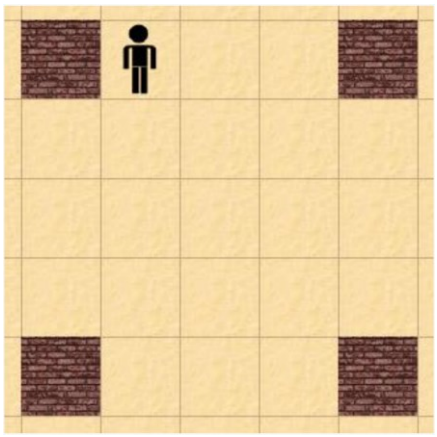
Erweitern Sie den Code der act()-Methode, um den Spieler nach Erreichen der unteren und linken Ränder der Welt richtig zu drehen.

3.4. Hinzufügen von Wänden

Schreiben Sie zwei neue Klassen als Unterklassen der Actor-Klasse. Die erste Klasse soll BrickWall sein (Ziegelmauer), und die zweite Klasse soll eine Wand (bspw. eine Holzwand) sein. Bereiten Sie geeignete 60x60 Pixelbilder in einem grafischen Editor vor. Dann ordnen Sie diese Bilder den neu erstellten Klassen zu.

3.5. Beobachtung der Bewegung des Spielers

Erstellen Sie vier Instanzen der BrickWall-Klasse und eine Instanz der Spielerklasse, wie in der Abbildung unten gezeigt. Raten Sie mal, wie sich der Spieler bewegen wird? Führen Sie die Anwendung aus. Stimmt Ihre Vorhersage mit dem überein, was Sie beobachten?



3.6. Kollisionserkennung bei Wänden

Erweitern Sie die Act()-Methode der Spielerklasse, um sicherzustellen, dass sich der Spieler gegen den Uhrzeigersinn 90° dreht, wenn er eine Zelle berührt, die eine Instanz der Wall-Klasse enthält.

In den folgenden Schritten wird das Verhalten der Spielerklasse geändert. Es wird eine vollständige Verzweigung und zusätzliche Bedingungen hinzugefügt. Überlegen Sie, wie sich eine Instanz der Spielerklasse verhält, wenn sie eine Zelle berührt, die eine Instanz der Wall-Klasse enthält.

3.7. Vorhersage des Bewegungsverlaufs eines Spielers

Überlegen Sie sich, wie sich eine Instanz der Spielerklasse bewegen wird. Stimmt das Ergebnis mit Ihrer Vorhersage überein?

3.8. Beobachtung der Randerkennung

Platzieren Sie eine Instanz der Spielerklasse in einer Ecke der Welt. Sagen Sie vorher, wie sich diese Instanz bewegen wird, wenn die Anwendung gestartet wird. Stimmt das Ergebnis mit Ihrer Vorhersage überein?

3.9. Vervollständigung der Mauer-Kollisionsbehandlung

Vervollständigen Sie die Kaskade von Bedingungen, so dass das Berühren einer Instanz der BrickWall-Klasse nur überprüft wird, wenn der Spieler nicht am Rande der Welt ist. Überprüfen Sie zuerst die Instanz der BrickWall-Klasse.

3.10. Automatische Bewegung des Spielers

Erstellen Sie eine Methode, um eine Instanz der Spielerklasse nur dann zu bewegen, wenn eine bestimmte Taste auf der Tastatur gedrückt gehalten wird, beispielsweise die m-Taste (m für move). Verschieben Sie alle Codezeilen aus der act()-Methode in die neue Methode. Die Methodenbezeichnung kann z.B. moveAutomatically() sein. Die Methode moveAutomatically muss dann innerhalb act() aufgerufen warden.

3.11. Bewegung des Spielers mit den Pfeiltasten

Erstellen Sie eine moveUsingArrows()-Methode in der Spieler-Klasse. Programmieren Sie diese Methode so, dass sich der Spieler nur bewegt, wenn eine Pfeiltaste gedrückt wird. Das Objekts soll sich in Richtung des gerdrückten Pfeils bewegen. Achten Sie darauf, den Code effizient zu halten. In der act()-Methode, rufen Sie die neue Methode auf.

3.12. Vorbereitung von Bildern

Bereiten Sie vier Bilder vor, die den Spieler in seiner Bewegung nach oben, unten, links und nach rechts zeigen. Die Abmessungen der Bilder dürfen die Abmessungen der Zelle, in unserem Fall 60x60 Pixel, nicht überschreiten. Legen Sie die vorbereiteten Bilder im Bildverzeichnis.

3.13. Einbau der Bilder enstprechend der Bewegungsrichtung

Erstellen Sie eine UpdateImage()-Methode, die das Image des Spielers gemäß seiner aktuellen Rotation einstellt. Fügen Sie einen Aufruf dieser Methode zum Körper der act()-Methode hinzu. Sie können auch das switch-Konstrukt zum Verzweigen zu verwenden, als Alternative zu if-else.

3.14. Ausführen der Greenfoot-Anwendung

Starten Sie die Anwendung. Prüfen Sie, ob sich die Bilder anpassen, sich drehen nach der Drehung des Spielers. Bearbeiten Sie Ihr Programm solange, bis es funktioniert.

3.15. Mehrere Spieler

Versuchen Sie, mehrere Spieler zu erstellen und sie mit der Tastatur zu steuern. Was beobachten Sie?

4. Variablen, Ausdrücke und fortgeschrittene Steuerung der Spieler-Objekte

Dieses Thema befasst sich mit der Einführung von Variablen in Form von Klassenattributen und Methodenparametern. Die Schüler verwenden die Attribute, um die Geschwindigkeit des Spielers zu steuern und bestimmte Keyboad-Tasten zu bestimmen, die die Bewegung eines Spielers steuern. Dies ermöglicht es, mehrere Spieler mit unterschiedlichen Keyboard-Tasten steuern zu lassen.

4.1. Erkennen von Unterschieden im Java-Code

Worin besteht der Unterschied zwischen den Code-Ausschnitten 1 und 2?

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. Schreiben einfacher Ausdrücke

Schreiben Sie einen Ausdruck, der boolesche und relationale Operatoren verwendet, um auszudrücken, dass eine Integer-Variable a einen Wert hat, wertmässig zu den folgenden Intervallen gehört: [-10,10), (5,142), (-11,-3) ODER (1,25].

4.3. Auswerten von Ausdrücken

Legen Sie Werte der Variablen x und y fest und fügen Sie sie zu den Ausdrücken hinzu. Was sind die Werte der Variablen b1 und b2 in den jeweiligen Codeausschnitten (1 und 2)?

1.

```
int x, y;
// ... assign values to x and y

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

2.

```
int x, y;
// ... assign values to x and y
```

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

In den nächsten Schritten ist die Spielerklasse zu ändern. Es sollen String-Attribute upKey, downKey, rightKey und leftKey hinzugefügt werden. Dies werden die Tastaturzeichen sein, die verwendet werden, um die Bewegung einer Instanz der Spielerklasse zu stützen. Das Keyword privat bedeutet, dass die Attribute nur innerhalb der Spielerklasse verfügbar sind. Innerhalb dieser Klasse werden die Attribute über das Wort dieser (d.h. eine Instanz dieser Klasse) aufgerufen, d.h. this.upKey, this.downKey, this.rightKey, und this.leftKey.

Im weiteren wird die moveUsingArrows()-Methode geändert, indem es die "left"-Tastangabe durch das this.left-Attribut ersetzt. Die anderen Schlüssel werden in ähnlicher Weise ersetzt. Die Schlüssel zur Steuerung der Bewegung einer bestimmten Instanz müssen angegeben werden, wenn die Instanz erstellt wird. Um dies zu tun, muss ein Konstruktor geschaffen werden. Dies ist eine spezielle Methode, um eine Instanz einer bestimmten Klasse zu schaffen. Die Konstruktor-Methode wird ausgeführt, wenn das Objekt erstellt wird. Der Konstruktor hat wieder vier Parameter - upKey, downKey, rightKey und leftKey. Im Inneren des Konstruktors müssen Sie zwischen leftKey und this.leftKey unterscheiden. Ersteres ist ein Parameter des Konstruktors und letztere ist ein Attribut der Instanz der Klasse. Der Konstruktor ist in der Klasse nicht zwingend. Wenn wir ihn nicht implementieren, dann wird ein Standardkonstruktor mit leerem Code verwendet.

4.4. Test des Konstruktors

Testen Sie den Konstruktor und die modifizierte Methode zur Steuerung der Spielerbewegung, indem Sie zwei Instanzen der Spielerklasse in die Welt einfügen. Für jede Instanz im Dialog sollten Sie verschiedene Keyboard-Tasten festlegen, um ihre Bewegung zu steuern. Testen Sie das Programm, um zu sehen, ob die eingesetzten Spieler unabhängig voneinander gesteuert werden können.

4.5. Umbenennen einer Klasse

Benennen Sie die MyWorld-Klasse in die Arena-Klasse um. Beachten Sie, dass der Konstruktorname der gleiche sein muss wie der Klassenname. Testen Sie, wie sich die Greenfoot-Umgebung verhält, wenn dies nicht derselbe ist.

Bisher war es immer notwendig, die Spieler-Objekte manuell hinzuzufügen, die nach dem Klick auf Reset auch wieder verschwunden sind. Das liegt daran, dass keine Instanzen im Konstruktor der Arena-Klasse angelegt werden. Im folgenden Schritt wird ein Spieler-Objekt von der Arena-Klasse erstellt. Wenn es privat ist, ist es nur innerhalb der Arena-Klasse verwendbar. Es wird dann mit dem new-Operator und mit der AddObject-Methode an der angegebenen Position erstellt. Bei der Esrtellung eines Objekts mit new, wird der Konstruktor der zugehörigen Klasse ausgelöst und das Objekt initialisiert.

4.6. Hinzufügen eines weiteren Spielers

Fügen Sie der Welt einen weiteren Spieler hinzu, zum Beispiel als Player2-Objekt, das von der linken Seite des Keyboards gesteuert wird, von den w-, s-, a- und d-Tasten für die Richtungen hoch, herunter, links und rechts. Platzieren Sie den Spieler an den Koordinaten [24,14]. Wenn das Objekt hinzugefügt wurde, ist die Steuerung zu testen.

4.7. Referenzen in Java

Was würde passieren, wenn wir nach der Erstellung der beiden Objekte die Zuweisung this.player1 = this.player2 anweisen würden? Wäre das ein Problem?

4.8. Erweiterung der Spieler-Klasse

Erweitern Sie die Spielerklasse mit einem zusätzlichen Attribut des Typs, der die Schrittgröße des Spielers darstellt. Hier kommen zwei Konstruktur-Methoden zum Einsatz, die sich in der Anzahl der Parameter unterscheiden. Das ist der sogenannte überladene Konstruktor.

4.9. Berücksichten der Schrittweite

Ändern Sie die moveUsingKeys()-Methode, um die Schrittweite zu berücksichtigen. Erstellen Sie eine neue Instanz der Spielerklasse und testen Sie die Funktionalität des Programms.

4.10. Bewegen der Spieler mit unterschiedlichen Geschwindigkeiten

Die nächste Aufgabe ist, zu programmieren, dass sich jeder Spieler entlang der Zellen bewegt, aber jeweils mit unterschiedlichen Geschwindigkeiten. Jeder Spieler kann eine andere Geschwindigkeit haben. Beispielsweise können Sie verschiedene Bewegungsgeschwindigkeiten programmieren, indem Sie den Player nicht jedes Mal bewegen, wenn ein Tastendruck erkannt wird (die Erkennung erfolgt in der act()-Methode, wenn Sie also eine Taste gedrückt halten, damit erkennen Sie das Tastendruck-Ereignis jedes Mal, wenn act() ausgeführt wird), aber nur bei jedem N. mal, wenn sie ausgeführt wird. Je größer N, desto niedriger wird die Geschwindigkeit des Spielers sein. Wie übermittelt man N an das Objekt und wo wird der Wert gespeichert? Woher erfährt man, wie viele Tastenanschläge verstrichen sind? (Hinweis: Es wird dafür ein Zähler benötigt).

5. Kooperation von Objekten und Klasen

Im Mittelpunkt dieses Themas steht die Objektzusammenarbeit. In diesem Thema fügen die Schüler dem Projekt die Interaktion zwischen Objekten in der Arena hinzu – zum Beispiel, um sicherzustellen, dass der Spieler nicht durch die Wände gehen kann. Darüber hinaus werden die Studenten in diesem Thema dem Projekt auch ein Bombenobjekt – einen der Hauptteile des Bomberman-Spiels – hinzufügen.

5.1. Programmieren der vertikalen Bewegung

Analog zur horizontalen Bewegung fügen Sie den Code für die Up- und Down-Anweisungen zu. Damit ändern Sie den Wert der lokalen Variable y.

Lokale Variablen sind für die weitere Vervollständigung des Programms wichtig, beispielsweise als Parameter der canEnter()-Methode.

5.2. Überprüfen der Bewegungsmöglichkeit

Ändern Sie die Methode, die die Bewegung des Spielers umsetzt (moveUsingArrows), um die Möglichkeit zu überprüfen, die Zielzelle zu betreten, bevor die Position des Spielers geändert wird.

Im daruffolgenden Schritt wird die Methode canEnter() erstellt.

5.3. Beachten von Ziegelmauern

Ändern Sie die canEnter()-Methode so, dass der Spieler auch auf Instanzen der Klasse BrickWall reagiert und diese nicht durchlaufen kann.

5.4. Hinzufügen einer Bombe

Erstellen Sie eine neue Bombenklasse, und entwerfen Sie ihre Attribute so, dass sie die Kraft der Explosion darstellen. Erstellen Sie einen parametrierbaren Konstruktor, und initialisieren Sie die Attribute des Objekts.

Mit dem folgenden Zwischenschritt wird der Spieler-Klasse ein Tastaturzeichen zum Platzieren von Bomben hinzugefügt. Außerdem wird ein Attribut für die Bombenleistung der Instanz des Bombenplatzierungsobjekts hinzugefügt: bombPower. Somit ist auch eine Modifikation des Konstruktors erforderlich.

5.5. Prüfen, ob das Bombenlegen möglich ist

Fügen Sie der Player-Klasse eine canPlantBomb()-Methode mit einem Rückgabewert vom Typ boolean hinzu, die ein Flag zurückgibt, das angibt, ob es möglich ist, eine Bombe auf der Zelle zu platzieren, in der sich der Spieler gerade befindet. Eine Bombe kann platziert werden, wenn die entsprechende Taste gedrückt wird und sich keine andere Bombe auf der Zelle befindet.

Mit dem folgenden Zwischenschritt werden der Bomb-Klasse die timer- und owner-Attribute hinzugefügt. Der Timer stellt die Zeit dar, die es dauert, bis die Bombe explodiert. Daher muss der Konstruktor der Bomb-Klasse geändert werden. Die act()-Methode ändert das Verhalten der Bombe. Wenn der Timer abläuft, explodiert die Bombe und verschwindet aus der Welt. Darüber hinaus wird die Player-Klasse geändert. Ein bombCount-Attribut wird hinzugefügt, um die Anzahl der Bomben darzustellen, die eine Instanz der Player-Klasse verwenden kann. Wenn eine Bombe platziert wird, verringert sie sich um 1. Wenn eine Bombe explodiert, erhöht sie sich um 1. Dies wird von der Methode bombExploded() gehandhabt.

5.6. Einbinden von Sound-Effekten

Erweitern Sie das Spiel, damit die Bombenexplosion von einem Soundeffekt begleitet wird. Der Ton kann aufgenommen oder aus dem Internet heruntergeladen werden. Suchen Sie sich dazu eine geeignete Methode aus der Greenfoot-Klassendokumentation heraus.

6. Vererbung und for-Schleife

In diesem Thema geht es um die Einführung der Vererbung. Die Schülerinnen und Schüler erstellen eine Superklasse für die Klassen Wall und BrickWall. Dann erstellen sie eine Testarena als Unterklasse der Arena-Klasse. Schließlich konzentriert sich dieser Abschnitt auf Schleifen mit einer festen Anzahl von Wiederholungen.

6.1. Hinzufügen einer übergeordneten Klasse

Erstellen Sie eine Klasse für ein Hindernis (Klassenname Obstacle). Welche Klasse ist die Superklasse der Hindernisklasse? Bearbeiten Sie die Kopfzeilen der Klassen BrickWall und Wall so, dass sie Unterklassen der Obstacle-Klasse sind.

6.2. Vereinfachung des Tests zum Betreten einer Zelle

Nach dem Hinzufügen der Hindernis-Superklasse ist es einfacher zu testen, ob der Spieler eine bestimmte Zelle betreten kann. In ihrer getObjectsAt()-Methode benötigt die Welt als dritten Parameter eine Klasse, nach der in einer bestimmten Zelle gesucht werden soll. Da sowohl BrickWall als auch Wall Obstacle sind, ist es möglich, sie einheitlich zu behandeln. Ändern Sie die canEnter()-Methode in der Player-Klasse so, dass nur eine einzige Liste von Hindernissen verwendet wird.

Der folgende Zwischenschritt behandelt die Erstellung der TestArena-Klasse als Unterklasse der Arena-Klasse.

6.3. Ändern der Arena-Klasse

Ändern Sie die Arena-Klasse so, dass ihr Konstruktor über zwei Parameter verfügt, die Breite und Höhe darstellen. Ändern Sie den Aufruf des übergeordneten Klassenkonstruktors in der Arena-Klasse, um diese Parameter zu übernehmen. Beachten Sie, dass die Arena nicht automatisch von Greenfoot erstellt werden kann, da sie Parameter für den Konstruktor entfernt. Entfernen Sie von diesem Konstruktor den Code, der für das Erstellen und Platzieren von Spielern in der Arena verantwortlich ist, dies wird von Unterklassen durchgeführt. Sie können auch die Deklaration von Attributen vom Typ Player aus der Arena-Klasse entfernen.

6.4. Korrigieren der TestArena-Klasse

Ändern Sie den Konstruktor der TestArena-Klasse, um eine leere Arena mit einer Größe von 7x7 Zellen zu erstellen. Um dies zu überprüfen, erstellen Sie eine Instanz der Klasse TestArena – wählen Sie aus dem Kontextmenü der Klasse TestArena den Eintrag new TestArena().

6.5. Hinzufügen einer Dimensionsabfrage

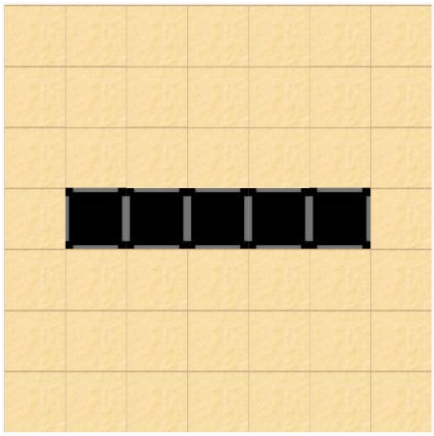
Fügen Sie der TestArena-Klasse eine showDimensions()-Methode hinzu, die die Arenaabmessungen auf dem Bildschirm druckt.

6.6. Hinzufügen einer weiteren Dimensionsabfrage

Fügen Sie der Player-Klasse eine showDimensions()-Methode hinzu, die die Abmessungen der Arena anzeigt, wenn sie sich in der Testarena befindet – Klasse TestArena. Verwenden Sie die showDimensions()-Methode der TestArena-Klasse.

6.7. Hinzufügen von Wänden zur Testarena

Ändern Sie den Konstruktor der TestArena-Klasse, um eine 7x7-Zellen-Arena mit Wänden zu erstellen, die wie folgt angeordnet sind:



Erinnern Sie sich daran, dass wir die Methode addObject() der World-Klasse verwenden können, die über drei Parameter verfügt, um eine Instanz der Actor-Klasse in die Welt einzufügen (d.h. in die Nachkommen der World-Klasse und in unserem Fall der Arena-Klasse):

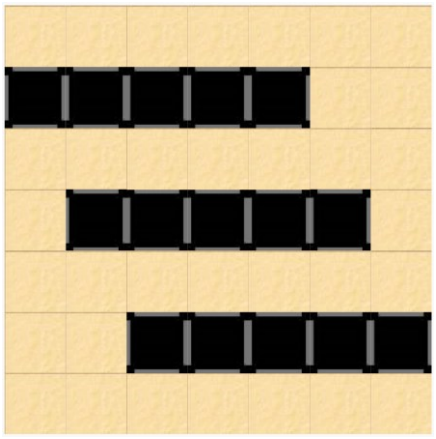
- einen Akteur, der eingefügt werden soll,
- x-Koordinate der einzufügenden Zelle (d.h. Spaltenindex von 0 nummeriert),
- y-Koordinate der einzufügenden Zelle (d.h. der Zeilenindex ab 0)

Die [0; 0] Position in der Welt befindet sich oben links. Da wir fünf Instanzen der Wall-Klasse auf einmal hinzufügen möchten, verwenden wir eine for-Schleife, die die Anweisungen innerhalb des Blocks für

alle i=1,2,3,4,5 in unserem Fall wiederholt. Denken Sie daran, dass super ein Aufruf des Konstruktors der übergeordneten Klasse ist, in unserem Fall der World-Klasse. Super muss an erster Stelle im Konstruktor stehen

6.8. Hinzufügen weiterer Wände

Ändern Sie den Konstruktor der TestArena-Klasse so, dass die Zellen wie in der Abbildung dargestellt angeordnet sind.



6.9. Überlegung, wie eine Reihe von Wänden dargestellt wird

Denken Sie darüber nach, welche Informationen benötigt werden, um eine Reihe von aufeinanderfolgenden Wänden erstellen zu können.

6.10. Hinzufügen einer Methode zum Erstellen einer Reihe von Wänden

Erstellen Sie eine createRowOfWalls()-Methode in der Arena-Superklasse, die über drei Parameter verfügt:

- die Zeile (die oberste Zeile hat den Index 0), in der mit dem Erstellen von Wänden begonnen werden soll,
- die Spalte (die linke Spalte hat den Index 0), von der aus mit dem Erstellen von Wänden begonnen werden soll,
- Eine Zahl, die angibt, wie viele aufeinanderfolgende Wände erstellt werden sollen.

Die Methode hat keinen Rückgabewert (daher verwenden wir das Schlüsselwort void).

Commit: b155b177619e41170ac7759f8d43ebf748a5da6b

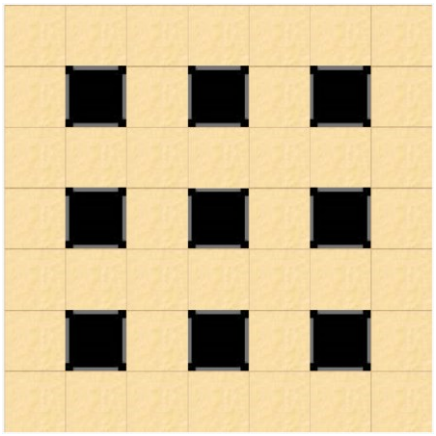
6.11. Verwenden der neuen Methode

Ändern Sie den Code im Konstruktor der Klasse TestArena, um die Methode createRowOfWalls() aus ihrer Oberklasse – der Klasse Arena – zu verwenden.

Commit: 91d8888d11fa710c0c2f7eb8a6e1c9da32139a66

6.12. Ändern der Anordnung der Wände

Ändern Sie den Konstruktor der TestArena-Klasse, um die in der folgenden Abbildung gezeigte Arena zu erstellen. Ändern Sie dazu die createRowOfWalls()-Methode so, dass ein vierter Parameter den Abstand zwischen den Wänden definiert.



6.13. Überlegen Sie, wie Sie ein Rechteck aus Wänden darstellen können

Überlegen wir, wie viele und welche Art von Informationen wir benötigen, um Wände in einer rechteckigen Anordnung erstellen zu können, deren Ausgangspunkt angegeben werden kann und für die der Abstand zwischen den Wänden sowohl in Zeilen als auch in Spalten festgelegt werden kann.

6.14. Hinzufügen einer Methode zum Erstellen eines Rechtecks aus Wänden

Deklarieren Sie die createRectangleOfWalls()-Methode im Arena-Vorgänger, der über die folgenden Parameter verfügt:

- die Zeile (die oberste Zeile hat den Index 0), von der aus mit dem Erstellen von Wänden begonnen werden soll,
- die Spalte (die linke Spalte hat den Index 0), von der aus mit dem Erstellen von Wänden begonnen werden soll,
- die Anzahl der zu erstellenden Zeilen,
- die Anzahl der aufeinanderfolgenden Wände, die in der Zeile erstellt werden sollen,
- die Anzahl der leeren Zellen (Zeilen) zwischen den Zeilen,
- Die Anzahl der leeren Zellen zwischen den Wänden in der Zeile.

Die Methode hat keinen Rückgabewert.

6.15. Verwenden der neuen Methode

Ändern Sie den Konstruktor der TestArena-Klasse so, dass er die Methode createRectangleOfWalls() verwendet, um die Arena wie in Aufgabe 6.12 gezeigt zu gestalten.

6.16. Test des Spiels

Erstellen Sie mehrere Wände in Ihrer Arena und fügen Sie zwei Spieler hinzu. Testen Sie die Funktionalität des Spiels, d.h. ob die Spieler nicht in einen BrickWall oder eine Wand gehen.

7. Listen und Programmschleifen (für jeweils)

Dieses Kapitel konzentriert sich im Detail auf Listen, die wir verwenden, um andere Objekte in der Arena zu verfolgen. Es führt die grundlegenden Methoden zur Arbeit mit einer Liste ein (Erzeugen, Hinzufügen eines Elements, Entfernen eines Elements, Zugriff auf ein Element) und beschreibt auch die Anwendung einer Programmschleife, um nacheinander auf alle Elemente einer Liste zuzugreifen.

7.1. Spielende prüfen

Erstellen Sie eine Methode isGameEnded() ohne Parameter in der Arena-Klasse, die erkennt, ob das Spiel beendet ist (es sind nur noch ein oder keine Spieler übrig) und im Rückgabewert des booleschen Typs angibt, ob es passiert ist. Gehen wir vorerst davon aus, dass das Ende des Spiels nie eintritt.

7.2. Beenden Sie das Spiel

Fügen Sie der Arena-Klasse die act()-Methode hinzu. Überprüfen Sie in der Methode, ob das Spiel beendet wurde (mit der Methode isGameEnded()) und wenn ja, stoppen Sie das Spiel. Um die Greenfoot-Umgebung zu stoppen, verwenden Sie Greenfoot.stop(); Befehl.

7.3. Hinzufügen einer Liste von Spielern

Fügen Sie der Arena-Klasse das Attribut listOfPlayers vom Typ LinkedList<Player> hinzu. Vergessen Sie nicht, dass Sie das Paket mit der LinkedList-Klasse importieren müssen. Initialisieren Sie das Attribut im Konstruktor der Arena-Klasse.

7.4. Registrieren Sie die Spieler

Fügen Sie der Arena-Klasse die Methode registerPlayer() hinzu, die einen einzelnen Parameter vom Typ Player akzeptiert und ihn am Ende der Liste von listOfPlayers mit der add()-Methode einfügt. Ändern Sie die Unterklassen der Arena-Klasse, um den Spieler in der Superklasse (Arena) zu registrieren, wenn ein Spieler an der richtigen Stelle in die Welt eingefügt wird.

7.5. Aufheben der Registrierung und Entfernen eines Spielers

Fügen Sie der Arena-Klasse, die einen einzelnen Parameter vom Typ Player akzeptiert, unregisterAndRemovePlayer() hinzu. Die Methode entfernt den Spieler aus der Spielerliste und entfernt ihn dann aus der Welt.

7.6. Beenden Sie das Spiel korrekt

Implementieren Sie den Text der isGameEnded()-Methode so, dass die Methode true zurückgibt, wenn ein oder kein Spieler mehr im Spiel ist. Verwenden Sie geeignete Listenmethoden.

7.7. Bomben gefährlich machen

Ändern Sie den Code in der act()-Methode der Bomb-Klasse so, dass die Bombe, bevor sie aus der Welt entfernt wird, alle Spieler erwirbt, die höchstens eine Kraft von ihr entfernt sind. Die getObjectsInRange()-Methode gibt eine Liste vom Typ List zurück. Sein erster Parameter ist der Bereich - in diesem Fall der Kraftmodifikator. Der zweite Parameter ist identisch mit der Klasse, deren Instanz innerhalb des angegebenen Bereichs durchsucht wird

7.8. Betroffene Spieler entfernen

Verwenden Sie die for-Schleife, um über alle Spieler in der Liste der von der Bombe betroffenen Spieler zu iterieren. Heben Sie die Registrierung solcher Spieler in der Arena-Klasse auf.

Der folgende Zwischencommit zeigt eine effizientere Möglichkeit, die Liste der betroffenen Spieler zu durchlaufen.

7.9. Bearbeiten der Spielerklasse

Erstellen Sie eine hit()-Methode in der Player-Klasse, die von der Bombe des Spielers aufgerufen wird, nachdem die Bombe ihn getroffen hat. Heben Sie bei dieser Methode die Registrierung des Spielers

von der Welt auf. Bearbeiten Sie den Code in der Methode act() der Bomb-Klasse, um die neue Funktionalität widerzuspiegeln.

7.10. Entfernen des Eigentümers

Erstellen Sie eine removeOwner()-Methode in der Bomb-Klasse, die ihr owner-Attribut auf null setzt. Eine Objektinstanz kann man sich als einen Zeiger vorstellen - einen "Pfeil" auf das Objekt. Wenn Sie den Zeiger auf null setzen, zeigt der Zeiger auf kein Objekt.

Im Moment haben wir das owner-Attribut auf null gesetzt. Das Attribut muss also nicht auf eine Instanz des Objekts verweisen. Daher müssen wir vor dem Aufruf der Methode bombExploded() überprüfen, ob ihr Besitzer existiert. Die Situation wird im folgenden Zwischencommit behoben.

7.11. Eine Liste von Bomben hinzufügen

Erstellen Sie ein Attribut listOfActiveBombs vom Typ LinkedList<Bomb> in der Player-Klasse. Initialisieren Sie es im richtigen Konstruktor. Ändern Sie die Methodentexte gemäß den folgenden Regeln:

- in der Methode act() eine neu erstellte Bombe in der listOfActiveBombs registrieren;
- Entfernen Sie in der Methode bombExploded() die Bombe, die als Parameter kam (diejenige, die explodiert ist) aus der listOfActiveBombs;
- Verwenden Sie in der hit()-Methode die for each-Schleife, um den Besitzer von allen Bomben in listOfActiveBombs zu entfernen.

8. Private Methoden und while-Schleife

In diesem Thema wird die while-Schleife vorgestellt – eine Schleife, die wiederholt wird, solange eine am Anfang der Schleife definierte Bedingung als "true" ausgewertet wird. Darüber hinaus lernen die Schülerinnen und Schüler auch, wie sie private Methoden erstellen – Methoden, die nur von einer Instanz der Klasse aufgerufen werden können, in der die Methode definiert ist.

8.1. Erstellen der Feuerklasse

Erstellen Sie die Fire-Klasse. Wählen Sie eine geeignete grafische Darstellung aus. Der Konstruktor dieser Klasse verfügt über einen Parameter, der bestimmt, wie lange das Feuer an Ort und Stelle brennt. Stellen Sie sicher, dass das Feuer nach einer bestimmten Zeit aus der Welt verschwindet.

8.2. Zünden Sie das Feuer an

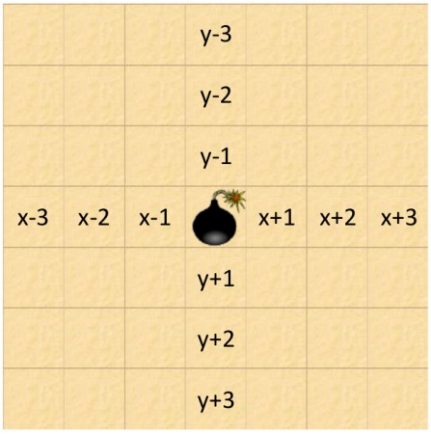
Ändern Sie den vorhandenen Bombenexplosionscode, um eine Instanz der Fire-Klasse am Bombenstandort zu belassen. Testen Sie Ihre Lösung.

8.3. Verbreiten Sie das Feuer

Verwenden Sie die for-Schleife, um die Bombenexplosion in die richtige Richtung von der Bombe aus zu verlängern (eine Instanz der Fire-Klasse zu erstellen). Dehne die Explosion auf so viele Zellen aus, wie durch das Stärkeattribut der Bombe angezeigt wird.

8.4. Verteilen Sie das Feuer in alle Richtungen

Passe die Bombenexplosion so an, dass sie Feuer in alle Richtungen erzeugt. Helfen Sie sich selbst, indem Sie die Koordinaten ändern, wie in der Abbildung unten gezeigt.



8.5. Schreibe die Loops neu

Schreiben Sie alle for-Schleifen für die Brandausbreitung mit der while-Schleife neu. Lassen Sie den zweiten Teil der Bedingung (es ist möglich, in der nächsten Zelle Feuer zu legen) vorerst weg.

Im folgenden Zwischencommit wird die Methode spreadFire() mit zwei Parametern erstellt. Gehen Sie die Methode durch und beschreiben Sie, wo das Feuer abhängig von den Eingabeparametern gesetzt wird. Die Methode ist privat. Das bedeutet, dass es nur aus der Bomb-Klasse aufgerufen werden kann.

8.6. Verwenden der privaten Methode

Verwenden Sie die private Methode spreadFire(), um das Feuer nach einer Bombenexplosion zu verbreiten.

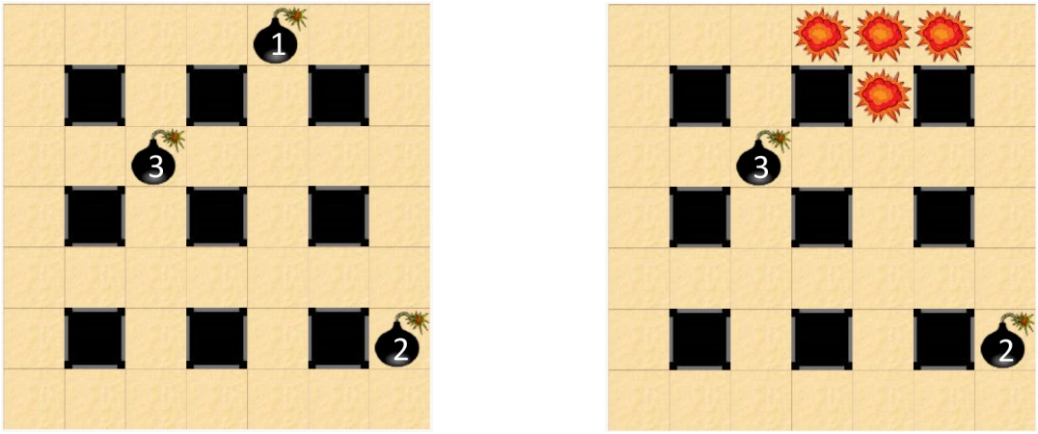
8.7. Hinzufügen einer weiteren privaten Methode

Erstellen Sie eine private Methode canCellExplode() in der Bomb-Klasse, die Zeilen- und Spaltenkoordinaten als Parameter verwendet und true zurückgibt, wenn in dieser Zelle eine Explosion auftreten kann, andernfalls false. Das Feuer kann nicht fortgesetzt werden, wenn:

- es erreichte den Rand der Welt,
- , wenn die Zelle eine Wand enthält.

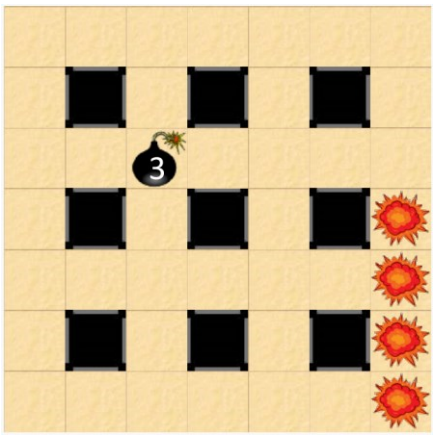
8.8. Verwenden der privaten Methoden

Mit der Methode canCellExplode() können Sie nun die Bedingung in der while-Schleife in der Methode spreadFire() in der Bomb-Klasse ändern. Ändern Sie die Bedingung in der Schleife, um das Ergebnis der Überprüfung von der canBombExplode()-Methode zu berücksichtigen. Testen Sie die Funktionalität der Lösung mit unterschiedlich starken Bomben zwischen den Wänden. Die Tests verschiedener Explosionen sind in den folgenden Abbildungen dargestellt:

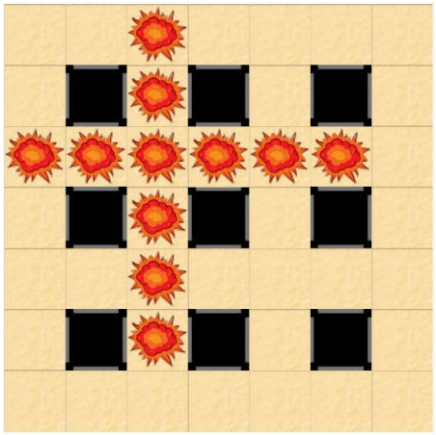


ein

b



c



d

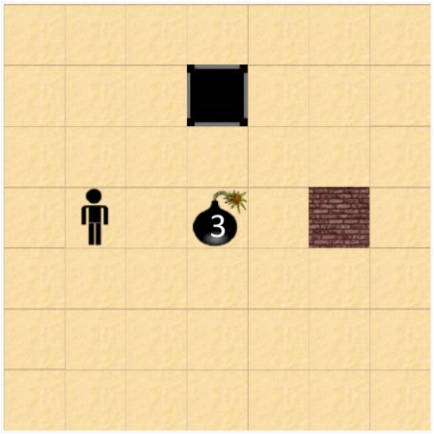
In Teil (a) sehen wir die ursprüngliche Verteilung, in Teil (b) explodierte eine Bombe mit Stärke 1, in Teil (c) explodierte eine Bombe mit Kraft 2 und in Teil (d) explodierte eine Bombe mit Kraft 3.

8.9. Explosionshindernis prüfen

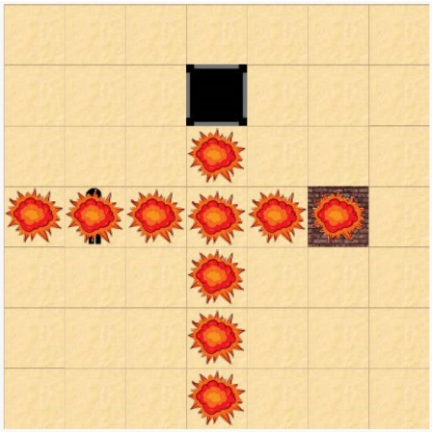
Fügen Sie der Klasse Bomb die private Methode canExplosionContinue() hinzu, die Zeilen- und Spaltenkoordinaten in Parametern akzeptiert und true zurückgibt, wenn die Zelle die Explosion nicht an den angegebenen Koordinaten gestoppt hat. Wenn die Zelle die Auflösung gestoppt hat, gibt die Methode false zurück. Die Explosion kann nicht fortgesetzt werden, wenn sie gegen eine Wand geprallt ist.

8.10. Begrenzen Sie die Explosion

Ändern Sie mit der canExplosionContinue()-Methode die spreadFire()-Methode in der Bomb-Klasse. Wenn die Auflösung von der angegebenen Zelle aus fortgesetzt werden kann, erhöhen Sie den Wert der Variablen i um 1 und berechnen Sie die Koordinaten der neuen Zeile und Spalte der Auflösung neu. Andernfalls erhöhen Sie den Wert der Variablen i künstlich auf einen Wert, der größer als die Kraft der Bombe ist, wodurch die Schleife gestoppt wird. Testen Sie Ihre Lösung anhand der folgenden Abbildung für die Situation:



Ein



b

Wie Sie auf dem Bild sehen können, explodiert die Ziegelwand, stoppt aber die Explosion. Es sollte also nach der Explosion verschwinden. Dies wird durch den folgenden Zwischencommit gelöst.

8.11. Ändern der Brandanwesenheitsprüfung

Ändern Sie das Verhalten der Bomb-Klasseninstanz so, dass sie die hit()-Methode des Spielers in ihrer Reichweite nicht aufruft. Stattdessen wird der Spieler selbst nach überlappendem Feuer suchen. Ändern Sie das Verhalten der act()-Methode des Players so, dass sie zuerst prüft, ob sie sich mit einer Instanz der Fire-Klasse überschneidet. Wenn dies der Fall ist, ruft es seine hit()-Methode selbst auf. Dadurch wird sichergestellt, dass der Spieler auch von dem Feuer getroffen wird, das nach der Explosion der Bombe brennt.

8.12. Füge eine Kette von Explosionen hinzu

Ändern Sie die act()-Methode der Bomb-Klasse so, dass die Bombe auch dann explodiert, wenn sie sich in derselben Zelle wie das Feuer befindet. Verifiziere die Lösung, indem du eine Kettenreaktion aus mehreren Bomben durchführst.

9. Polymorphismus

Das Ziel dieses Themas ist es, den Schülern beizubringen, wie sie virtuelle Methoden erstellen und sie bei Bedarf in Unterklassen überlagern können. Außerdem werden die Schülerinnen und Schüler mit einer neuen Sichtbarkeit für Attribute und Methoden vertraut gemacht – dem Modifikator für den geschützten Zugriff. Die Schülerinnen und Schüler werden Polymorphismus nutzen, um bestehenden Code zu vereinfachen.

9.1. Minen hinzufügen

Beginnen wir mit dem Hinzufügen einer Mine. Die Mine explodiert, sobald der Spieler sie betritt. Eine Mine explodiert auch immer, wenn sie von Feuer getroffen wird (z. B. von einer Bombe, die in der Nähe explodiert ist). Er hinterlässt das Feuer an seinem Platz (und nur dort). Es wurde die Möglichkeit hinzugefügt, dass der Spieler Minen (wie Bomben) legen kann, wenn eine Taste gedrückt wird (z. B. Strg oder Umschalt). Ähnlich wie bei Bomben hat der Spieler auch eine begrenzte Anzahl an Minen (d.h. wenn er alle Minen legt, kann er nur dann eine weitere Mine legen, wenn eine der zuvor gelegten Minen explodiert). Die anfängliche Anzahl der Minen wird durch einen Parameter des Konstruktors des Spielers festgelegt. Um Minen zu registrieren und auf ihre Explosion in der Player-Klasse zu reagieren, gehen Sie wie bei Bomben vor (erstellen Sie eine Liste von Minen, fügen Sie die Methoden mineExploded(), canPlantMine() usw. hinzu).

9.2. Oberklasse hinzufügen

Erstelle eine gemeinsame Superklasse für die Klassen Bombe und Mine – die Klasse Explosiv. Welche Attribute und Methoden sollen in die Oberklasse verschoben werden, welche sollen in den Unterklassen verbleiben? Ändern Sie vorhandene Klassen entsprechend Ihrem Entwurf.

9.3. Anpassen der Sichtbarkeit von Attributen

Ändern Sie die Sichtbarkeit des Besitzerattributs in der Explosive-Klasse in protected. Die Sichtbarkeit des privaten Attributs würde es ermöglichen, es nur innerhalb der Explosive-Klasse und nicht in ihren Nachkommen zu verwenden. Geschützt bedeutet Sichtbarkeit innerhalb des Pakets, das in unserem Fall Teil des Bomberman-Projekts ist.

9.4. Hinzufügen einer Textausgabe

Erstellen Sie eine printWhoYouAre()-Methode ohne Parameter in der Explosive-Klasse, die keinen Rückgabewert hat. Die Methode gibt den Text "EXPLOSIVE" auf dem Bildschirm aus, auf dem sich der Sprengstoff gerade befindet. Erstellen Sie eine Instanz der Mine-Klasse und rufen Sie die printWhoYouAre()-Methode auf. Was ist los? Erstellen Sie eine Instanz der Bomb-Klasse und rufen Sie die printWhoYouAre()-Methode auf. Was passiert in diesem Fall?

9.5. Verbessern Sie die Textausgabe

Erstellen Sie eine printWhoYouAre()-Methode in der Mine mit dem gleichen Header wie in der Klasse Explosive (d.h. die Methode hat den gleichen Namen, die gleichen Parameter und den gleichen Rückgabewerttyp). Die Methode gibt den Text "MINE" auf dem Bildschirm aus. Erstellen Sie erneut eine Instanz der Mine-Klasse und der Bomb-Klasse. Versuchen Sie zu erraten, was passiert, wenn Sie die Testmethode in einer Instanz der Klasse Mine und einer Instanz der Klasse Bomb aufrufen. Rufen Sie danach die Methoden auf. Stimmt Ihre Vermutung mit dem Ergebnis überein?

9.6. Abschließen der Textausgabe

Überschreiben Sie die printWhoYouAre()-Methode in der Bomb-Klasse, sodass sie den Text "BOMB" auf den Bildschirm schreibt. Überprüfen Sie die Richtigkeit Ihrer Lösung.

9.7. Explosionsbehandlung hinzufügen

Erstellen Sie die Methoden shouldExplode() und explosion() in der Klasse Explosive und die Methode explosiveExploded() in der Klasse Player, wie oben beschrieben. Implementieren Sie die Methodenkörper noch nicht. Wenn ein Rückgabewert benötigt wird, geben Sie false zurück.

9.8. Lass den Sprengstoff explodieren

Schreiben Sie den Text der act()-Methode in die Explosive-Klasse.

9.9. Lass die Bombe explodieren

Überschreiben Sie die Methoden shouldExplode() und explosion() in der Bomb-Klasse. Verwenden Sie den entsprechenden Code aus der act()-Methode. Beachten Sie, dass es möglich ist, die Körper der Methoden einfach zu schreiben, da es nicht notwendig ist, über die Bedingungen nachzudenken (die Superklasse hat dies getan).

So wie super für den Konstruktor verwendet wurde, um den übergeordneten Konstruktor aufzurufen, ist dies auch für andere Methoden möglich. Der folgende Zwischencommit zeigt einen Aufruf der act()-Methode in der Bomb-Klasse von der Explosive-Klasse mit super.

9.10. Lass die Mine explodieren

Überschreiben Sie die Methoden shouldExplode() und explosion() in der Mine-Klasse. Verwenden Sie den entsprechenden Code aus der act()-Methode. Warum ist es notwendig, die act()-Methode am Ende zu entfernen?

9.11. Interaktion mit Feuer hinzufügen

Ändern Sie den Text der shouldExplode()-Methode in der Explosive-Klasse so, dass die Methode true zurückgibt, wenn die Instanz eine Instanz der Fire-Klasse berührt. Ändern Sie die überschriebenen Methoden shouldExplode() in der Bomb-Klasse und der Mine-Klasse, um die Funktionalität der Vorgängermethode zu verwenden.

9.12. Vereinfachen Sie Spielerattribute

Entfernen Sie die Attribute listOfActiveBombs und listOfActiveMines aus der Player-Klasse. Fügen Sie der Player-Klasse ein einzelnes Attribut vom Typ listOfActiveExplosives vom Typ LinkedList<Explosive> hinzu. Initialisieren Sie es im Konstruktor, und entfernen Sie die Initialisierung der ursprünglichen Attribute aus dem Konstruktor.

Im Zusammenhang mit der vorherigen Änderung muss die Player-Klasse überall dort geändert werden, wo die Listen listOfActiveBombs und listOfActiveMines verwendet wurden. Sie müssen durch die listOfActiveExplosives-Listen ersetzt werden. Alles wird im folgenden Zwischencommit gelöst.

9.13. Vereinfachen Sie die Handhabung von Explosionen

Implementieren Sie den Hauptteil der Methode explosiveExploded(). Verwenden des instanceof-Operators, um zu bestimmen, ob es sich bei dem Sprengstoff um eine Bombe oder um eine Mine handelt. Basierend auf dem tatsächlichen Typ wird der Zähler der verfügbaren Bomben oder der Zähler der verfügbaren Minen erhöht. Achten Sie darauf, den Sprengstoff aus der Liste der aktiven Sprengstoffe zu entfernen. Entfernen Sie schließlich die unnötigen Methoden bombExploded() und mineExploded().

10. Zufallszahlen

Dieses Thema widmet sich der Zufälligkeit. Die Schülerinnen und Schüler lernen die Zufallsklasse kennen. Mit Hilfe seiner Instanzen generieren sie Zufallszahlen. Das Thema zeigt auch eine Möglichkeit, Zufallszahlen zu generieren, ohne die Random-Klasse zu verwenden, indem die Greenfoot-Umgebung direkt verwendet wird. Die Schüler verwenden Zufallszahlen, um das Layout der Arena zufällig zu gestalten und der Welt Boni hinzuzufügen – spezielle Elemente, die nach der Explosion der Ziegelmauer erstellt werden und die Eigenschaften ausgewählter Spieler verbessern.

10.1. Denken Sie an den Zufall

Denken Sie darüber nach, was Zufall ist, wie wir ein zufälliges Ergebnis aus einem Experiment erhalten können und welche zufälligen Phänomene wir in der Welt um uns herum beobachten.

10.2. Denken Sie an die Generierung von Zufallswerten

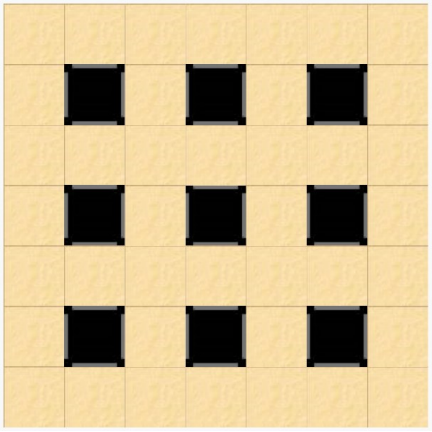
Betrachten wir einen klassischen Würfel mit sechs Seiten. Könnten wir es verwenden, um eine zufällige Position auf einem Schachbrett mit 6x6 Feldern zu erzeugen? Wie wäre es mit einem Schachbrett mit 3x3 Feldern? Wie würde sich die Methode der Generierung ändern, wenn wir eine Münze verwenden würden? Schlagen Sie solche Algorithmen zur Positionsgenerierung vor.

10.3. Zufälligkeit beobachten

Generieren Sie mit dem Algorithmus aus der vorherigen Aufgabe und mit Hilfe eines Würfels zufällige Positionen auf dem Schachbrett. Notieren Sie Ihre Ergebnisse auf dem Schachbrett. Kann eine Regelmäßigkeit in den Ergebnissen beobachtet werden?

10.4. Bereite eine zufällige Arena vor

Bereite eine Arena vor. Erstellen Sie eine Unterklasse der Arena-Klasse, die Sie z.B. RandomArena nennen. Legen Sie die entsprechende Weltgröße im Konstruktor fest. Wir empfehlen ein regelmäßiges Wandlayout mit einem leeren Feld zwischen den Wänden, wie in der folgenden Abbildung dargestellt:



Im Zwischencommit ist die leere Methode createRandomWall() vorbereitet.

10.5. Hinzufügen eines Zufallszahlengenerators

Fügen Sie der RandomArena-Klasse ein Verweisattribut vom Typ Random hinzu. Beachten Sie, dass die Klasse Random im Paket java.util.Random definiert ist. Initialisieren Sie das Attribut im Konstruktor.

Im folgenden Zwischencommit werden Zufallszahlen generiert und in den Variablen randomColumn und randomRow gespeichert.

10.6. Überprüfen der Zellenbelegung

Fügen Sie der RandomArena-Klasse eine private isCellFree()-Methode hinzu, die zwei Parameter annimmt – column und row. Die Methode gibt true zurück, wenn die Zelle in der Welt frei ist (keine Instanz der Klasse Actor enthält), andernfalls gibt sie false zurück.

Nun ist es möglich, die Koordinaten einer Zelle zu generieren, in der es kein Objekt der Klasse Actor gibt, und eine Instanz der Klasse BrickWall in diese Zelle einzufügen. Alles wird durch die while-Schleife im folgenden Zwischencommit gelöst.

10.7. Zufällige Wände generieren

Ändern Sie den Konstruktor der RandomArena-Klasse so, dass Wände nach dem Zufallsprinzip in einem Drittel aller Zellen in der Arena generiert werden.

10.8. Verschieben Sie die Zufälligkeit auf den Vorfahren

Verschieben Sie die Methoden createRandomWall(), isCellFree() und das Generatorattribut (einschließlich seiner Initialisierung im Konstruktor) in die Superklasse Arena. Vergessen Sie nicht, auch die Importzeilen zu verschieben.

10.9. Generalisieren der Zufallsgenerierung

Fügen Sie der Methode createRandomWall() einen Parameter vom Typ Actor hinzu – dies ist der Akteur, den wir an zufälligen Koordinaten einfügen werden. Ändern Sie den Methodennamen (z. B. insertActorRandomly()) und aktualisieren Sie den Methodenaufruf aus der RandomArena-Klasse.

10.10. Boni hinzufügen

Erstellen Sie eine Bonus-Klasse als Unterklasse der Actor-Klasse. Erstelle zwei Unterklassen der Bonusklasse – die Klasse BonusFire und die Klasse BonusBomb. Legen Sie geeignete Bilder für die Klassen fest.

10.11. Boni nach dem Zufallsprinzip erstellen

Bearbeiten Sie den Code in der act()-Methode der BrickWall-Klasse. Nachdem sie zerstört wurde, wird an ihrer Stelle mit einer Wahrscheinlichkeit von 10 % ein Bonusfeuer und mit einer Wahrscheinlichkeit von 10 % eine Bonusbombe erzeugt. In 80% der Fälle entsteht nach der Vernichtung nichts.

10.12. Wenden Sie den Bonus an

Bereite die Bonusklasse vor. Erstellen Sie eine geschützte applyYourself()-Methode ohne Rückgabewert, die einen einzelnen Parameter vom Typ Player akzeptiert. Lassen Sie diese Methode in der Bonus-Klasse leer. Definieren Sie dann eine Aktion in der act()-Methode, um zunächst zu erkennen, ob ein Spieler auf den Bonus getreten ist (Methode (Player)this.getOneIntersectingObject(Player.class)) und wenn ja, wenden Sie sie an (durch Aufrufen der Methode applyYourself()) und entfernen Sie schließlich den Bonus aus der Welt.

10.13. Erhöhen Sie die Anzahl der Bomben

Fügen Sie der Player-Klasse eine öffentliche increaseBombCount()-Methode ohne Parameter hinzu, die keinen Rückgabewert hat und die Anzahl der Bomben, die der Spieler platzieren kann, um eine erhöht. Überschreiben Sie dann die applyYourself()-Methode in der BonusBomb-Klasse. Das Anwenden des Bonus bedeutet, die Anzahl der Bomben, die der Spieler hat, um eine zu erhöhen (Aufruf der Methode increaseBombCount()).

10.14. Erhöhen Sie die Bombenkraft

Fügen Sie der Player-Klasse eine öffentliche Methode increaseBombPower() ohne Parameter hinzu, die keinen Rückgabewert hat und den Wert des bombPower-Attributs um eins erhöht. Überschreiben Sie dann die applyYourself()-Methode in der BonusFire-Klasse und erhöhen Sie die Bombenstärke des Spielers um eins.

3.2. Projekt Turmverteidigung (Tower Defense)

In Tower-Defense-ähnlichen Spielen verwendet der Spieler Türme, die eine Art Kugel abschießen, um Feinde daran zu hindern, eine Kugel zu erreichen und zu zerstören. Die Feinde folgen immer dem gleichen Weg, aber im Laufe des Spiels werden die Feinde stärker und spawnen in größeren Gruppen. Der Spieler muss die Türme an strategischen Orten platzieren, um sie in kommenden Wellen zu stoppen. Es gibt viele Versionen des Spiels, die in verschiedenen Welten mit unterschiedlichen Entitäten stattfinden (von Ballons bis hin zu Orks, Verteidigung mit tierischen Türmen bis hin zu magischen Pools).

Dieses Projekt wird eine Art von Turm einführen, der vom Spieler manuell gesteuert werden kann oder nicht. Die Feinde sind von unterschiedlicher Art und unterscheiden sich in ihren HP und ihrer Geschwindigkeit. Das vorgestellte Spieldesign ist leicht zu erweitern, was genügend Raum für die Kreativität der Schüler sowie für die Aufgaben der Lehrer lässt. Aus diesem Grund haben wir in den ersten Kapiteln versucht, Aktivitäten vom Typ Bewertung zu minimieren. Darüber hinaus lässt das Design genügend Raum, um Themen außerhalb des Rahmens von Light OOP (wie z. B. Polymorphismus) mit Leichtigkeit einzuführen. Das Projekt in seinem endgültigen Zustand während des Spiels wird in Abbildung 3.

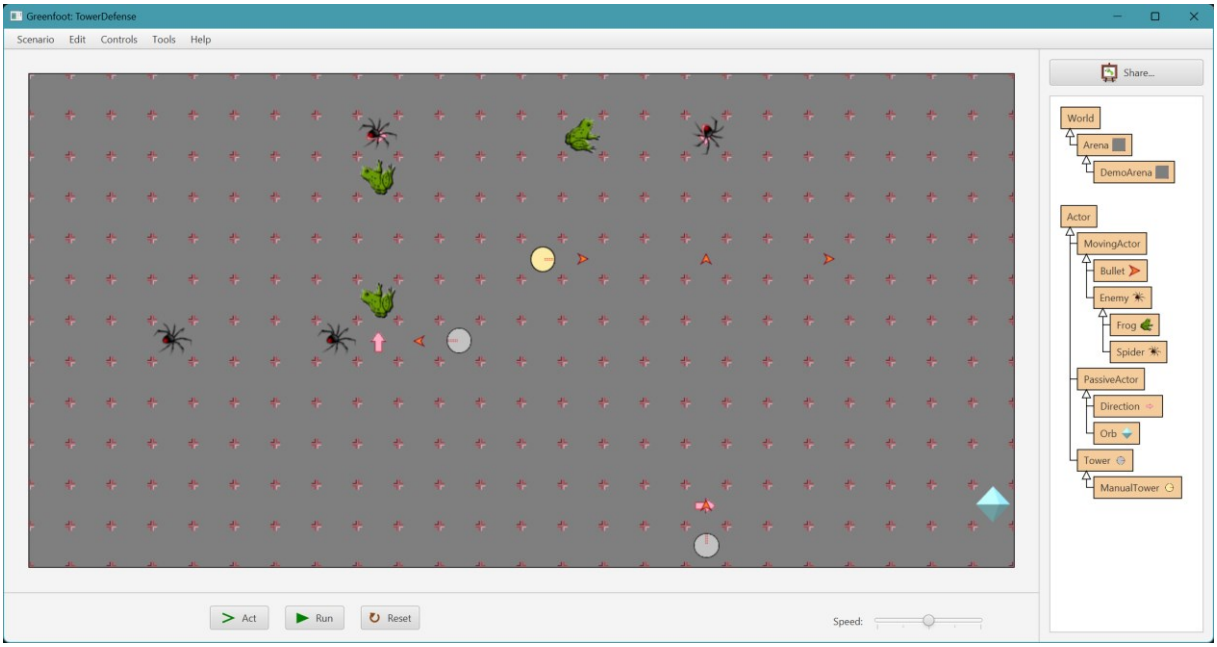


Abbildung 3: Greenfoot-Umgebung mit finalem Stand des Projekts Tower Defense

Der vollständige Quellcode ist verfügbar unter:
https://gitlab.kicon.fri.uniza.sk/oop4fun/project-tower-defense

Das Lerndesign kann über die folgende URL abgerufen warden::
http://learning-design.eu/en/preview/452257b563cbf14b6f06acfd/details

3.2.1. Inhalt und Umfang des Bildungsprogramms

Die Gesamtarbeitsbelastung der Lernenden beträgt 33 Stunden und 5 Minuten und verteilt sich wie folgt:

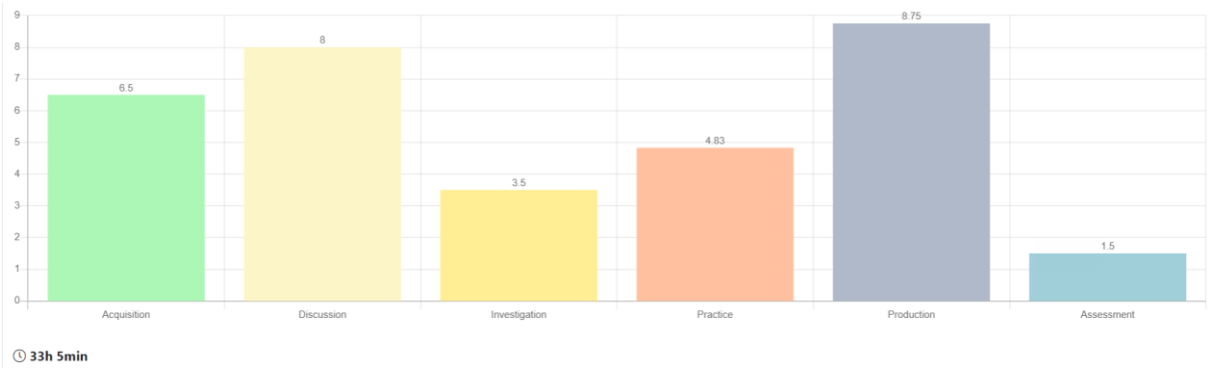


Abbildung 4: Arbeitsbelastung der Lernenden bei der Verwendung des Projekts Tower Defense

Die konstruktive Ausrichtung ist in der folgenden Tabelle zusammengefasst:

Tabelle 2: Konstruktive Ausrichtung des Projekts 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						

Einen detaillierten Plan finden Sie in Anlage 5.2.

3.2.2. Themen

Das Projekt Tower Defense gliedert sich in sieben Themenbereiche:

1. Einführung in die Greenfoot-Umgebung
2. Algorithmus, Anwendungssteuerung, Methodenerstellung
3. Verzweigung und Feindkontrolle
4. Variablen und Ausdrücke
5. Verband
6. Erbschaft
7. Verkapselung

Behandelte Themen von Light OOP sind:

- Klassen, Objekte, Instanz
- Methoden, Übergeben von Methodenargumenten
- Erbauer
- Attribute
- Statische Variablen und Methoden
- Verkapselung
- Erbschaft
- Abstrakte Klassen
- Objekt-Live-Zyklus

1. Einführung in die Greenfoot-Umgebung

Das Thema widmet sich der Projekterstellung. Die Schüler werden in der Lage sein, ein neues Projekt in der Greenfoot-Umgebung zu erstellen, eine Klasse (als Unterklasse von **`Actor) zu erstellen`**, ein Bild für eine neu erstellte Klasse auszuwählen, ihre Instanz zu erstellen und eine Nachricht an sie zu senden.

Erstellen Sie ein neues Projekt. Geben Sie ihm einen richtigen Namen (z. B. **`TowerDefense`**) und speichern Sie ihn an einem geeigneten Ort.

Tabelle 3 fasst den Vergleich der Arbeitslasten des Themas Greenfoot-Umgebung zwischen den Projekten Bomberman und Tower Defense zusammen. Es gibt keinen Unterschied in der Gestaltung der Themen.

Tabelle 3: Vergleich der Arbeitslasten des Themas Greenfoot-Umfeld zwischen den Projekten Bomberman und Tower Defense

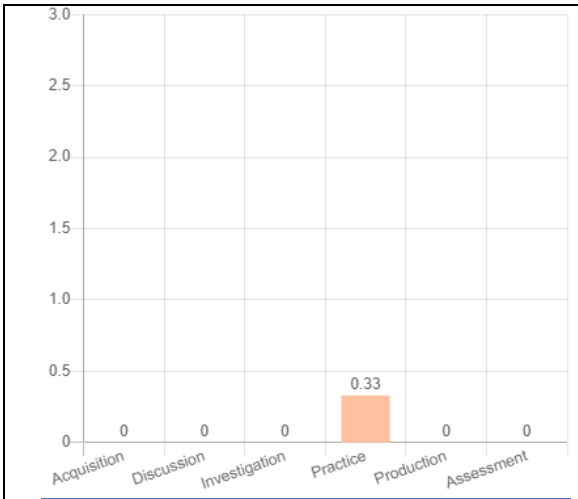
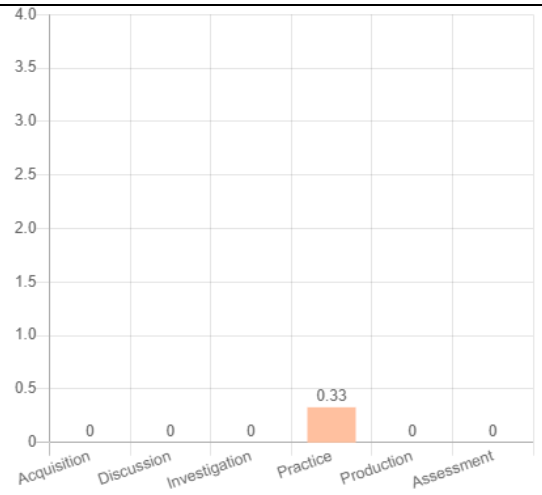
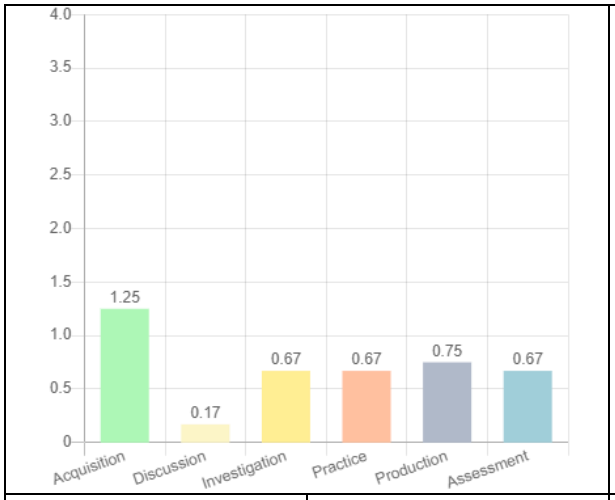
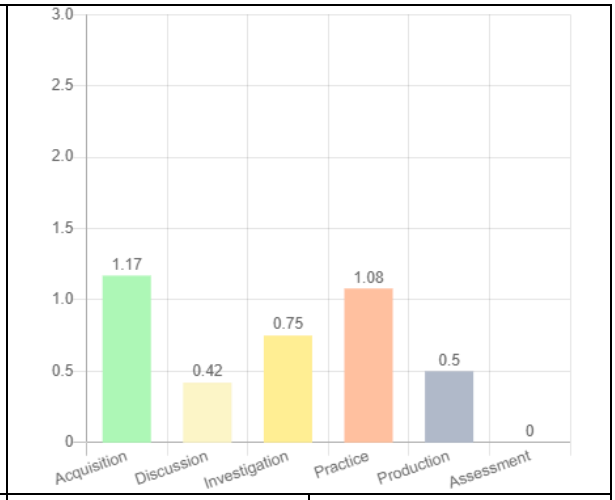
	Bomberman	0h 20min		Turmverteidigung	0h 20min

Tabelle 4 fasst den Vergleich der Arbeitslasten des Themas Klassendefinition zwischen den Projekten Bomberman und Tower Defense zusammen. Die Arbeitsbelastung des Tower-Defense-Projekts ist geringer, praxisorientierter mit Schwerpunkt auf Untersuchung und Praxis.

Tabelle 4: Vergleich der Arbeitslasten des Themas Klassendefinition zwischen den Projekten Bomberman und Tower Defense

	Bomberman	4h 10min		Turmverteidigung	3h 55min

1.1. Aufgabe 1.1 Identifikation von Objekten aus dem Projekt Projekt Bomberman.

1.2. Bereite die Welt vor

Bearbeiten Sie den Quellcode der Klasse `MyWorld` (doppelklicken Sie darauf), um eine Welt mit einer Größe von 24x12 Zellen zu erstellen. Jede Zelle sollte 50 Pixel groß sein.

1.3. Vorbereiten von Weltgrafiken

Finden oder erstellen Sie ein geeignetes Bild für den Welthintergrund. Sie können entweder vorbereitete Images verwenden (wählen Sie den Punkt `Bild setzen...` aus dem Kontextmenü der Klasse `MyWorld`) oder benutzerdefiniertes Bild (kopieren Sie das Bild in den Unterordner `images` Ihres Projektordners und wählen Sie es auf die gleiche Weise wie zuvor beschrieben aus).

Als Hintergrund können Sie ein einzelnes Bild verwenden, das die gesamte Weltfläche abdeckt (berechnen Sie die benötigte Größe des Bildes in Bezug auf die Größe der Welt) oder ein kleineres, das wiederholt kopiert wird (verwenden Sie ein quadratisches Bild mit der Größe der Zelle).

1.4. Erstelle die Klasse Enemy

Erschaffe einen Feind. Der Feind marschiert auf die Kugel des Spielers zu, um sie zu beschädigen und schließlich zu zerstören. Erstellen Sie eine neue Unterklasse der Klasse Actor (wählen Sie den Punkt `Neue Unterklasse...` aus dem Kontextmenü der Klasse `Actor`). Geben Sie ihm den richtigen Namen (`Feind`) und das Bild.

1.5. Instanz der Klasse Enemy erstellen

Erstellen Sie eine Instanz der Klasse Enemy (wählen Sie den Punkt `new Enemy()` aus dem Kontextmenü der Klasse `Enemy`, setzen Sie die Instanz in die Welt mit einem linken Mausklick an der gewünschten Position). Untersuchen Sie den internen Zustand (wählen Sie den Punkt `Prüfen` aus dem Kontextmenü der erstellten Instanz).

Erstelle eine weitere Instanz der **`Klasse Enemy`** und bringe sie an eine andere Position. Vergleichen Sie die internen Zustände von zwei erstellten Instanzen.

1.6. *Senden von Nachrichten an die Instanz*

Senden Sie eine Nachricht an die Instanz der Klasse `Enemy` (wählen Sie das vom **`Schauspieler geerbte Element`** aus dem Kontextmenü der ausgewählten Instanz aus und wählen Sie dann das gewünschte Element aus). Was ist passiert? Wie wurde der interne Zustand der jeweiligen Instanz beeinflusst?

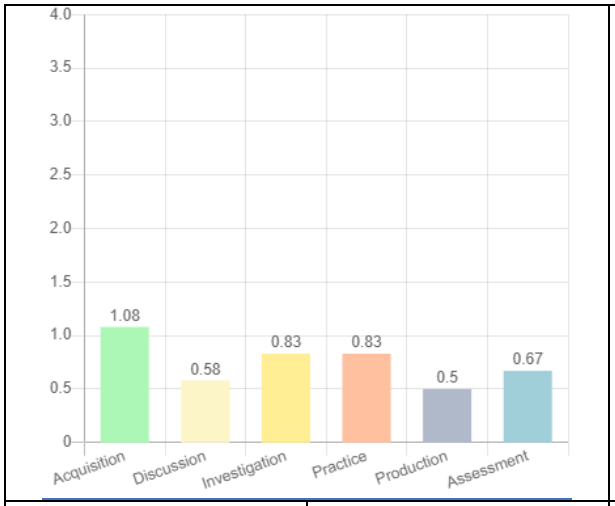
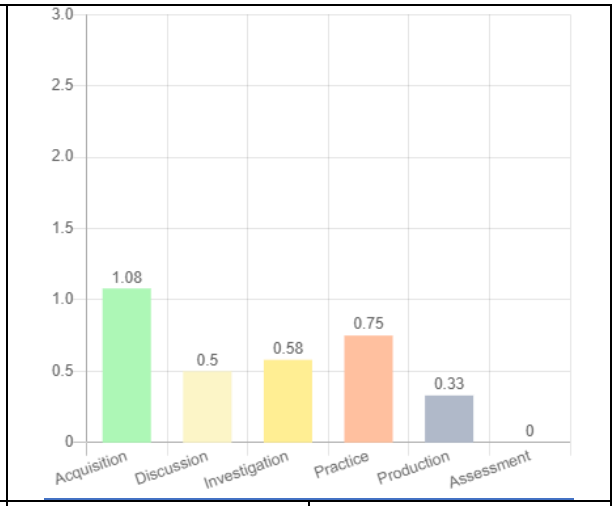
Sende Nachrichten an die Instanz der Klasse `Feind`, damit sie sich in Position [12, 6] bewegt und nach unten zeigt. Schreiben Sie die Reihenfolge der gesendeten Nachrichten auf Papier.

2. Algorithmus, Anwendungssteuerung, Methodenerstellung

Das Thema befasst sich mit den Grundlagen der Algorithmisierung und führt von Anfang an in die Arbeit mit der Dokumentation ein. Die Studierenden sind in der Lage, eine Methode im Quellcode aufzurufen, Dokumentation zu schreiben und aufzurufen.

Tabelle 5 fasst den Vergleich der Arbeitslasten des Themas Algorithmus zwischen den Projekten Bomberman und Tower Defense zusammen. Das Design von Tower Defense ähnelt dem von Bomberman, jedoch mit einer deutlich geringeren Anzahl von TLAs des Untersuchungstyps. Sehen Sie, dass viele TLAs die gleichen sind wie im Bomberman-Projekt. Dies liegt daran, dass es in diesem Zustand der Projekte eine große Überschneidung dessen gibt, was damit gemacht werden kann. Es ist möglich, sich von den Aufgaben des Bomberman-Projekts inspirieren zu lassen, wenn es notwendig ist, den Untersuchungsteil des Lehrplans zu stärken. Um jedoch leichte OOP mit Hilfe des Tower-Defense-Projekts zu unterrichten, halten wir die vorgeschlagene Menge an Ermittlungstyp-TLAs für ausreichend.

Tabelle 5: Vergleich der Arbeitslasten des Themas Algorithmus zwischen den Projekten Bomberman und Tower Defense

	Bomberman	4h 30min		Turmverteidigung	3h 15min

2.1. *Aufgabe **Error! Reference source not found. Error! Reference source not found.** aus dem Projekt Projekt Bomberman.*

2.2. *Aufgabe **Error! Reference source not found. Error! Reference source not found.** aus dem Projekt Projekt Bomberman.*

2.3. *Aufrufen einer Methode*

Fügen Sie dem Text der **`act()`-**Methode eine Anweisung hinzu , sodass die Instanz der **`Enemy-Klasse`** zwei Zellen in die aktuelle Richtung verschiebt. Erstellen Sie dann weitere Instanzen der **`Enemy-Klasse`**, und rufen Sie die Methode für jede Instanz auf. Wird das Verhalten erwartet?

2.4. Dokumentation hinzufügen

Fügen Sie einen Dokumentationskommentar für die **`act()`**-Methode hinzu.

2.5. Fügen Sie weitere Dokumentation hinzu

Bearbeiten Sie den Dokumentationskommentar der **`Enemy-Klasse. Fügen Sie die Version der Klasse und ihren Autor hinzu.`**

2.6. Aufgabe Error! Reference source not found. Error! Reference source not found. aus dem Projekt Projekt Bomberman

2.7. Aufgabe 2.9 Benutzen der Greenfoot-Anwendungssteuerung aus dem Projekt Projekt Bomberman

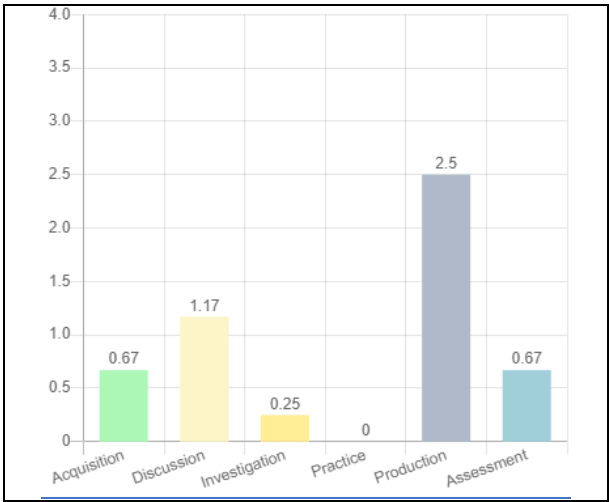
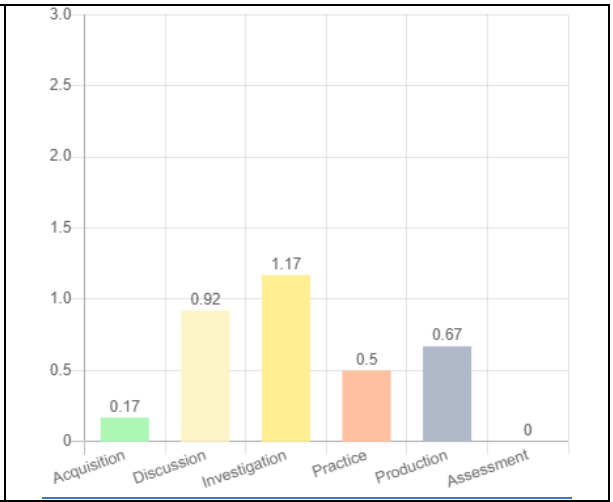
3. Verzweigung und Feindkontrolle

Das Thema behandelt unvollständige und vollständige Verzweigungen. Die Grundlagen der Wahrnehmung der **`Schauspielerwelt`** werden vorgestellt. Die Studierenden sind in der Lage, Code unter Verwendung von Bedingungen zu schreiben.

Der Stand des Projekts in diesem Kapitel eröffnet dem Lehrer die Möglichkeit, Aufgaben wie "Verwende die Instanzklasse von **`Direction`** und **`Orb, um`** den Feind **`so zu navigieren`**, dass er sich im gewünschten Muster bewegt" zuzuweisen, mit Limitbedingungen wie der maximalen Anzahl von Instanzen einer bestimmten Klasse oder Aufgaben wie "Bewegung in der gewünschten Anordnung der Welt vorhersagen".

Tabelle 6 fasst den Vergleich der Arbeitslasten des Themas Verzweigung zwischen den Projekten Bomberman und Tower Defense zusammen. Es gibt einen deutlichen Unterschied zwischen den Designs. Tower Defense teilt die Produktionstyp-TLAs auf, um die Ermittlungen und das Training zu stärken. Dies ermöglicht es, mehr zu experimentieren und lässt die Tür für die Kreativität der Schüler offen. Beachten Sie die geringe Anzahl von Akquisitions-TLAs. Dies liegt daran, dass a) keine Mehrfachverzweigung eingeführt wird und b) Untersuchungs-TLAs im Vordergrund stehen.

Tabelle 6: Vergleich der Arbeitslasten des Themas Algorithmus zwischen den Projekten Bomberman und Tower Defense

			
Bomberman	5h 15min	Turmverteidigung	3h 25min

3.1. Beobachte den feindlichen Zustand

Erstelle eine Instanz der Klasse **`"Gegner"`** und platziere sie in der Mitte des Spielbretts. Öffnen Sie ein Fenster mit dem internen Zustand der Instanz und positionieren Sie es so, dass es sichtbar ist,

während die Anwendung ausgeführt wird. Führen Sie dann die Anwendung aus, und beobachten Sie, wie sich die Werte der **`Attribute x`**, **`y`** und **`rotation`** in der **`Enemy-Klasse`** ändern. Wie ändern sich diese Werte, wenn Sie sich bewegen (nach oben, unten, links und rechts) und sich drehen?

3.2. Hinzufügen der Erkennung von Weltkanten

Fügen Sie dem Text der **`act()-Methode Code hinzu`** , um den Feind um 180° zu drehen, nachdem er den Rand der Welt erreicht hat.

3.3. Hinzufügen der Klassen Direction und Orb

Erstellen Sie zwei neue Klassen, die von der Actor-Klasse abstammen. Die erste Klasse wird **`die Richtungsklasse sein und die zweite Klasse wird`** Kugel sein. Bereiten Sie geeignete (max. 50x50 Pixel) Bilder in einem grafischen Editor vor. Ordnen Sie diese Bilder dann den neu erstellten Klassen zu.

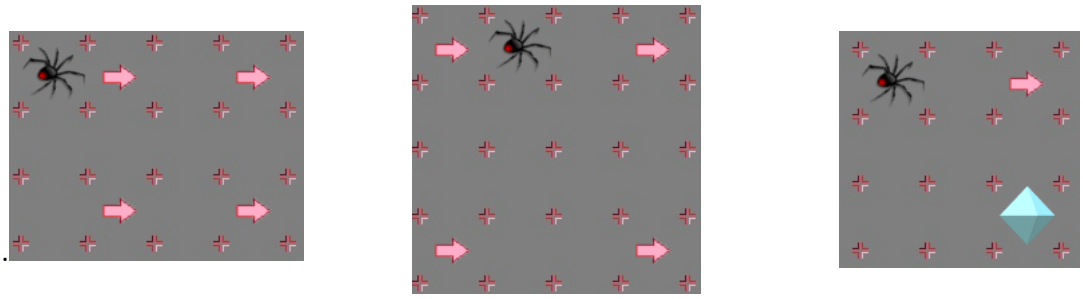
3.4. Kollisionserkennung hinzufügen

Fügen Sie der **`act()-`**Methode der **`Enemy-Klasse`** Code hinzu , um sicherzustellen, dass:

- der Feind dreht sich um 90° im Uhrzeigersinn, wenn er eine Zelle betritt, die eine Instanz der **`Klasse Direction`** enthält ,
- Der Feind dreht sich um 90° gegen den Uhrzeigersinn, wenn er eine Zelle betritt, in der sich eine Instanz der **`Orb-Klasse befindet.`**

3.5. Vorhersage der feindlichen Bewegung bei benutzerdefinierten Setups

Bereiten Sie verschiedene Konfigurationen vor, Inspiration finden Sie in den folgenden Abbildungen. Ratet mal, wie sich der Feind bewegen wird? Führen Sie die Anwendung aus. Stimmt Ihre Vorhersage mit dem überein, was Sie beobachten? Was verursachte Unterschiede in der Vorhersage und der Realität?

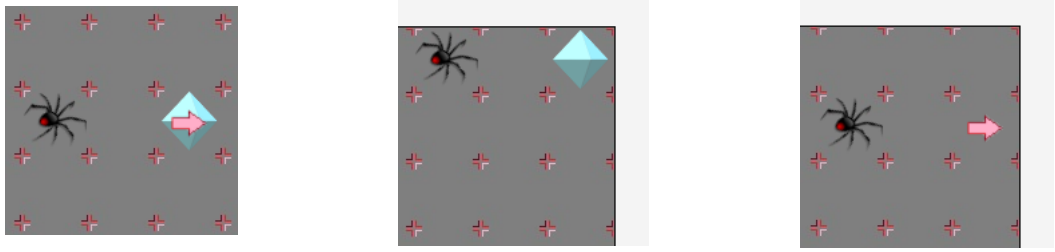


a) b) c)

Abbildung 5: Konfigurationen von benutzerdefinierten Setups von Instanzen zur Vorhersage der Bewegung von Instanzen der Klasse Enemy

3.6. Vorhersage der feindlichen Bewegung in einem bestimmten Setup

Bereiten Sie die Situation vor, wie in der Abbildung unten dargestellt. Ratet mal, wie sich der Feind bewegen wird? Führen Sie die Anwendung aus. Stimmt Ihre Vorhersage mit dem überein, was Sie beobachten? Was verursachte Unterschiede in der Vorhersage und der Realität?



a) b) c)

Abbildung 6: Konfigurationen von kniffligen Setups von Instanzen zur Vorhersage der Bewegung von Instanzen der Klasse Enemy

3.7. Verwenden Sie vollständige Verzweigung mit Kollisionserkennung

Im letzten Kapitel können Sie sehen, dass das Problem auftritt, wenn sich eine Instanz der **`Orb-`** oder Direction-Klasse am Rande der Welt befindet oder sich beide Instanzen in derselben Zelle befinden. Wenn die Verzweigung unvollständig ist, kommt es zu Kollisionen oder wiederholten Drehungen. Grundsätzlich sind mehrere Bedingungen gleichzeitig erfüllt. Daher müssen Sie den Code der act()-Methode der Enemy-Klasse so ändern, dass nur eine Umdrehung stattfindet. Es ist notwendig, eine vollständige Verzweigung zu verwenden. Erstellen Sie eine Kaskade von Bedingungen. Die wichtigste Prüfung (d. h. die erste) ist die Kantenerkennung. Die zweitwichtigste Prüfung ist die Touch-Prüfung für eine Instanz der **`Direction-Klasse`**. Die letzte ist die Prüfung, ob eine Instanz der Klasse **`Orb`** berührt wurde. Ändern Sie die **`Methode act()`** gemäß dieser Regeln ab.

3.8. Vorhersage der feindlichen Bewegung in früheren Setups

Wiederholen Sie die Schritte aus den Aufgaben 3.5 und 3.6. Was hat sich geändert?

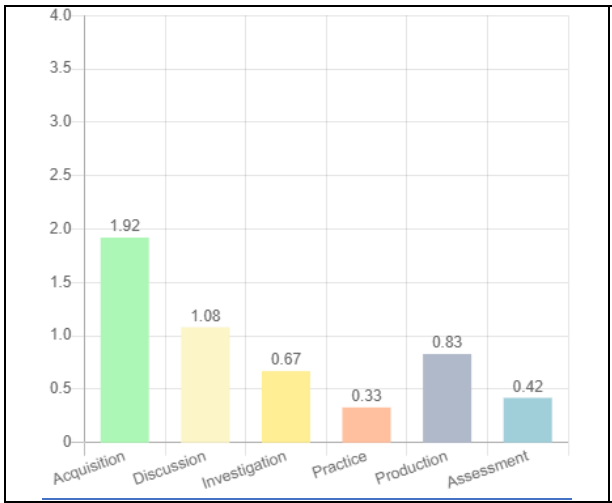
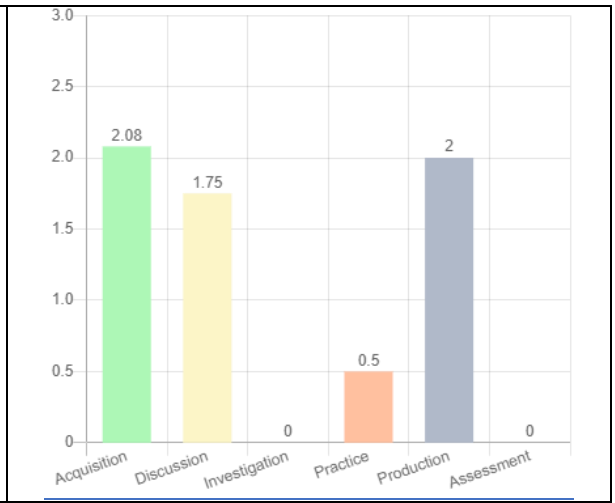
4. Variablen und Ausdrücke

In diesem Kapitel werden Variablen und Ausdrücke vorgestellt.

Der Stand des Projekts in diesem Kapitel eröffnet ähnliche Möglichkeiten wie im letzten Kapitel. Mit ein wenig Kreativität können weitere Klassen hinzugefügt werden, wie z.B. die Direction-Klasse, die je nach Instanz der Klasse Enemy ein anderes Verhalten beim Betreten (z.B. Teleporter, Tunnel, etc.) verursachen. Diese Lehrveranstaltungen können mit den Studierenden besprochen und die jeweilige Umsetzung als Hausaufgaben vergeben werden.

Tabelle 7 fasst den Vergleich der Arbeitslasten des Themas Variablen und Ausdrücke zwischen den Projekten Bomberman und Tower Defense zusammen. Sehen Sie die Ähnlichkeit zwischen den Entwürfen früherer und aktueller Themen zwischen den beiden Projekten. Dieses Thema ist mehr produktionsorientiert (ähnlich wie das vorherige Thema in Bomberman) und umgekehrt (beachten Sie, dass Bomberman eine andere Art von Projekt ist, bei dem die Kreativität der Schüler mit Vorteilen in diesem Thema genutzt wurde).

Tabelle 7: Vergleich der Arbeitslasten des Themas Variablen und Ausdrücke zwischen den Projekten Bomberman und Tower Defense

	Bomberman	5h 15min		Turmverteidigung	6h 20min

4.1. Drehen Sie sich in die Richtung

Ändern Sie den Code in **der** act()-**Methode von Enemy** so, dass er sich in die gleiche Richtung wie die **Direction-Klasseninstanz** dreht (sie haben die gleiche Rotation). Verwenden Sie **die Methode getOneIntersectingObject(_cls_),** um die Instanz in der richtigen lokalen Variablen zu speichern (**Richtungsrichtung** - Sie müssen Typecast verwenden, da der Rückgabewert vom Typ **Actor ist**, schreiben **Sie (Direction)** vor den Aufruf der **getOneIntersectingObject-Methode**, wir werden später zur Typumwandlung kommen). Wenn eine Instanz der entsprechenden Klasse abgerufen wurde, extrahieren Sie die Drehung der Richtung mit der **Methode getDirection()** (speichern Sie sie, wenn Sie möchten) und setzen Sie sie dann mit **der Methode** setDirection(int) **in enemy (this).** Testen Sie Ihre Lösung.

4.2. Umbenennung der Klasse MyWorld in Arena

Geben Sie der Klasse **MyWorld** einen besseren Namen. Benennen Sie es in **Arena um**. Vergessen Sie nicht, den Konstruktor entsprechend umzubenennen.

4.3. Layout der Arena erstellen

Erstellen Sie ein benutzerdefiniertes Layout der **Arena**. Füllen Sie den Konstruktor der Klasse aus. Füge eine Instanz von **Enemy**, eine Instanz von **Orb** und mindestens eine Instanz von **Direction hinzu**. Um (Unterklasse von) **Akteur hinzuzufügen**, können Sie die folgende Vorlage verwenden:

1. Deklarieren und Initialisieren der Variablen des erforderlichen Typs (Unterklasse von Actor)
   Feind e = neu Feind();
2. Zuweisen von Eigenschaften mit den richtigen Methoden
   e.setRotation(90);
3. Platzieren Sie es in der Welt (**Arena**) mit der Methode **addObject(Actor).**
   this.addObject(e, 6, 0);

Testen Sie Ihre Lösung.

4.4. Identifizieren Sie das Problem mit der Bewegung und schlagen Sie eine Lösung vor

Identifizieren Sie, was die Probleme mit der Bewegung verursacht. Wie können diese Probleme gelöst werden?

Der Feind** bewegt sich derzeit 2 Zellen auf einmal, was zu Problemen bei der Bewegung führt. Wir können die Geschwindigkeit des Feindes anders modellieren. Die Instanz des **Gegners** bewegt sich immer um 1 Zelle auf einmal. Wir führen jedoch eine Bewegungsverzögerung ein – die Instanz von **Enemy** bewegt sich, nachdem **die Verzögerungsaufrufe der Methode** act() **bestanden haben.

4.5. Attribut Enemy.moveDelay

Fügen Sie ein neues Attribut vom Typ **int** mit dem Namen **moveDelay** in die Klasse **Enemy ein**. Erstellen Sie einen parametrischen Konstruktor mit Parameter, um dieses Attribut zu initialisieren. Initialisieren Sie das Attribut mit dem Parameter. Passen Sie den Code in **Arena** entsprechend an.

4.6. Bewegung von Gegnern unter Berücksichtigung der Verzögerung

Aktualisieren Sie die Methode **act()** der Klasse **Enemy**, sodass sie nach **moveDelay-Aufrufen** der Methode verschoben wird. Führen Sie ein neues Attribut **nextMoveCounter ein**. Initialisieren Sie sie auf **0**. Ändern Sie **die act()**-Methode so, dass sie **this.move(1)** nur aufruft , wenn **nextMoveCounter** 0 **erreicht**. Setzen Sie nach der Verschiebung **nextMoveCounter** auf den Wert von **moveDelay zurück**. Wenn sich die Instanz von **Enemy** nicht bewegen konnte (weil **nextMoveCounter** nicht 0 **erreicht hat**), verringern Sie **nextMoveCounter** um **1**.

4.7. Parametrischer Konstruktor der Klasse Direction

Fügen Sie der Klasse Direction einen parametrischen Konstruktor mit einer alleinigen Parameterdrehung vom Typ **`int hinzu`**. Rotiert die erstellte Instanz im Hauptteil des Konstruktors um den Parameter. Passen Sie den Code in **`Arena`** entsprechend an.

4.8. Überladungskonstruktoren in der Klasse Direction

Überladen von Konstruktoren in der Klasse **`Direction`** durch Hinzufügen eines nicht-parametrischen Konstruktors. Rufen Sie im Hauptteil des nichtparametrischen Konstruktors den parametrischen Konstruktor mit der Argumentrichtung gleich **`0 auf`**. Passen Sie den Code in **`Arena`** entsprechend an – rufen Sie nach Möglichkeit die nicht-parametrische Version des Direction-Klassenkonstruktors auf.

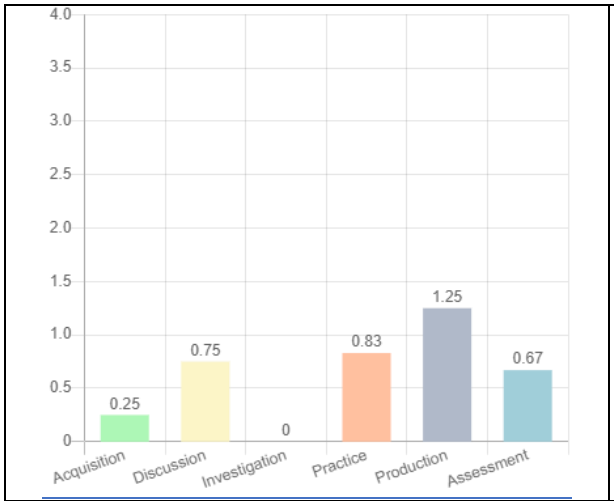
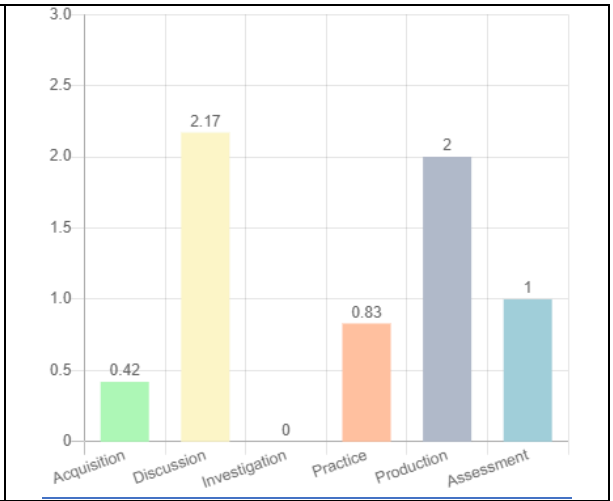
5. Verband

Das wichtigste Thema dieses Projekts ist der Verein. Die Diskussion wird genutzt, um herauszufinden, wie die Zusammenarbeit verschiedener Objekte zu komplexem Verhalten führen kann, obwohl die Codes in den Objekten leicht zu verstehen und zu pflegen sind. UML-Sequenzdiagramme werden verwendet, um die Zusammenarbeit von Objekten zu veranschaulichen und den Algorithmus auf kooperative Objekte zu verteilen. Dieses Diagramm kann während der Diskussion mit der Klasse erstellt werden.

Das Projekt befindet sich nach diesem Thema als abgeschlossen. In den folgenden Kapiteln wird die Variabilität der Anwendung erhöht, wobei der Schwerpunkt auf den Vorteilen von OOP bei richtiger Anwendung liegt.

Tabelle 8 fasst den Vergleich der Arbeitslasten des Themas Assoziation zwischen den Projekten Bomberman und Tower Defense zusammen. Wir betrachten das Verständnis von Assoziation als die wichtigste Kompetenz, wenn wir dieses Projekt für die Lehre von OOP nutzen. Daher haben wir die Produktion und die Diskussion von TLAs deutlich verstärkt. Beachten Sie, dass es auch Bewertungs-TLAs gibt. Diese sind so konzipiert, dass sie zuvor durchgeführte TLAs in einem etwas anderen Kontext verwenden.

Tabelle 8: Vergleich der Arbeitslasten des Themas Assoziation zwischen den Projekten Bomberman und Tower Defense

			
Bomberman	3h 45min	Turmverteidigung	6h 25min

5.1. Diskutiere, was passieren soll, wenn der Feind die Kugel erreicht.

Nachdem der Feind die Kugel erreicht hat, verringert die Kugel ihre LP. Sind die HP = 0, endet das Spiel, ansonsten wird der Feind in der Arena respawnt.

5.2. Besprechen, wie die Instanz der Klasse Feind mit den relevanten Objekten interagieren sollte, indem Nachrichten verwendet werden, wenn die Instanz der Klassenkugel getroffen wird

Der Algorithmus wird auf kooperierende Objekte verteilt.

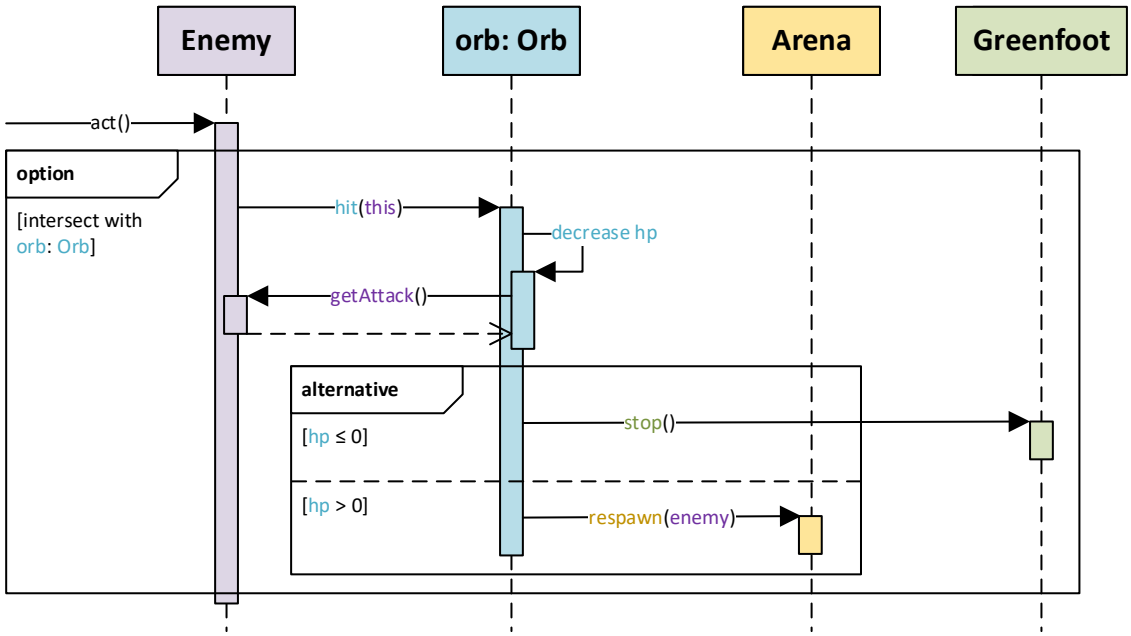


Abbildung 7: UML-Sequenzdiagramm der Instanz der Klasse Enemy, die mit anderen Objekten interagiert, wenn sie auf die Instanz der Klasse Orb trifft

5.3. Attribute Enemy.attack und Orb.hp

Fügen Sie ein neues Attribut vom Typ **`int`** mit dem Namen **`attack`** in der Klasse **`Enemy hinzu`**. Fügen Sie dem Konstruktor einen Parameter hinzu, um dieses Attribut zu initialisieren. Initialisieren Sie das Attribut mit dem Parameter.

Fügen Sie ein neues Attribut vom Typ **`int`** mit dem Namen **`hp`** in die Klasse **`Orb ein`**. Fügen Sie einen parametrischen Konstruktor mit einem Parameter hinzu, um dieses Attribut zu initialisieren. Initialisieren Sie das Attribut mit dem Parameter.

Passen Sie den Code in **`Arena`** entsprechend an.

5.4. Getter des Attributs Enemy.attack

Erstellen Sie einen Getter (Methode, die verwendet wird, um den Wert eines Attributs zu erhalten) des Attributangriffs in der Klasse **`Enemy`**.

5.5. Erstelle und teste die Methode Arena.respawn(Enemy)

Die Methode **`respawn`** ohne Rückgabewert und mit dem einzigen Parameter vom Typ **`Enemy`** in der Klasse **`Arena hinzugefügt`**. Legen Sie in der Methode die Position und Drehung der Instanz der Klasse **`Enemy`** auf die gleichen Werte fest, wie sie im Konstruktor erstellt wurden.

Testen Sie die Methode. Nachdem die Instanz von **`Arena`** erstellt wurde, starten Sie die Anwendung nicht. Ziehe stattdessen die Instanz von **`Enemy`**. Rufen Sie dann das Kontextmenü der **`Arena-Instanz auf (Sie müssen mit der rechten Maustaste in die Arena klicken, in der es keine`**

Instanz einer anderen Klasse gibt) und wählen Sie das Element **der Respawn-Methode**. Um einen Parameter auszufüllen, stellen Sie sicher, dass die Anwendung angehalten und mit dem Parameter "Aktiv abgelegt" abgelegt ist (wobei der Cursor darin blinkt). Wenn ja, klicke mit der linken Maustaste auf die Instanz von **Enemy**. Beobachten Sie, welcher Ausdruck in dem Fenster erstellt wurde. Klicken Sie dann auf die Schaltfläche OK und sehen Sie, was passiert.

5.6. Methode Orb.hit(Enemy) erstellen und testen

Fügen Sie der Methode **hit** ohne Rückgabewert und mit dem einzigen Parameter vom Typ **Enemy** in die Klasse **Orb hinzu**. Lass den Körper leeren.

Testen Sie den Aufruf der Methode. Führen Sie ähnliche Schritte wie oben aus, rufen Sie jedoch das Kontextmenü der Instanz der Klasse **Orb auf**. Beobachten Sie, welcher Ausdruck in dem Fenster erstellt wurde.

5.7. Rufen Sie die Methode Orb.hit(Enemy) von Enemy auf

Ändern Sie den Code in der Methode **act()** der Klasse **Enemy**, so dass die Methode **hit()** aufgerufen wird, wenn die Instanz von **Enemy** (**this**) auf die Instanz von **Orb trifft**.

Alte Codes entfernt, die dazu führten, dass sich der Feind drehte, wenn die Kugel getroffen wurde, und die dazu führten, dass der Feind vom Rand der Welt abprallte.

5.8. Implementierung der Methode Orb.hit(Enemy)

Implementieren des Hauptteils der Methode **Orb.hit (Feind)** in Bezug auf die im Rahmen der Aufgabe vorgenommene Analyse 5.2. Testen Sie Ihre Anwendung.

5.9. Hinzufügen der Klassen Bullet und Tower

Hinzufügen von Klassen **Bullet (Kugel, Geschoss)** und **Tower** (**Turm**). Verwenden Sie die gleichen Prinzipien wie in der Aufgabe 3.3.

Commit*:* ece4df70042c8f60098e14ad2cee55514897d825

5.10. Diskutiere, wie sich die Instanz der Klasse Kugel bewegen soll und was passieren soll, wenn sie die Instanz der Klasse Feind oder den Rand der Arena erreicht.

Die Kugel sollte sich bewegen, bis sie den Feind oder den Rand der Welt erreicht. Das Kugelzeichen ändert die Richtung der Bewegung nicht. Die Geschwindigkeit des Kugels kann mit dem gleichen Mechanismus wie in der Aufgabe gesteuert werden 4.6.

5.11. Implementieren der Verschiebung der Instanz der Klasse Bullet

Anwenden von Wissen, das in Aufgaben behandelt wird 3.2, 3.4 und 4.6. Platzieren Sie einen Dokumentationskommentar in den Code, wobei die Interaktion mit der Instanz der Klasse **Feind** passieren sollte.

5.12. Diskutiere, wie die Instanz der Klasse Tower die Instanz der Klasse Bullet abschießt.

*Denken Sie daran, dass die Instanz von **Turm** sollte nicht bei jedem Aufruf der Methode ausgelöst werden **handeln()**. Begeistern Sie sich durch den Mechanismus, der in 4.6. Trennen Sie relevante Schritte in Methoden der Klasse **Turm**.*

5.13. Besprechen Sie, wie die Instanz der Klasse Tower mit den relevanten Objekten interagieren sollte, indem Sie beim Schießen Nachrichten verwenden.

Verwenden Sie analoge Prinzipien wie in 5.2.

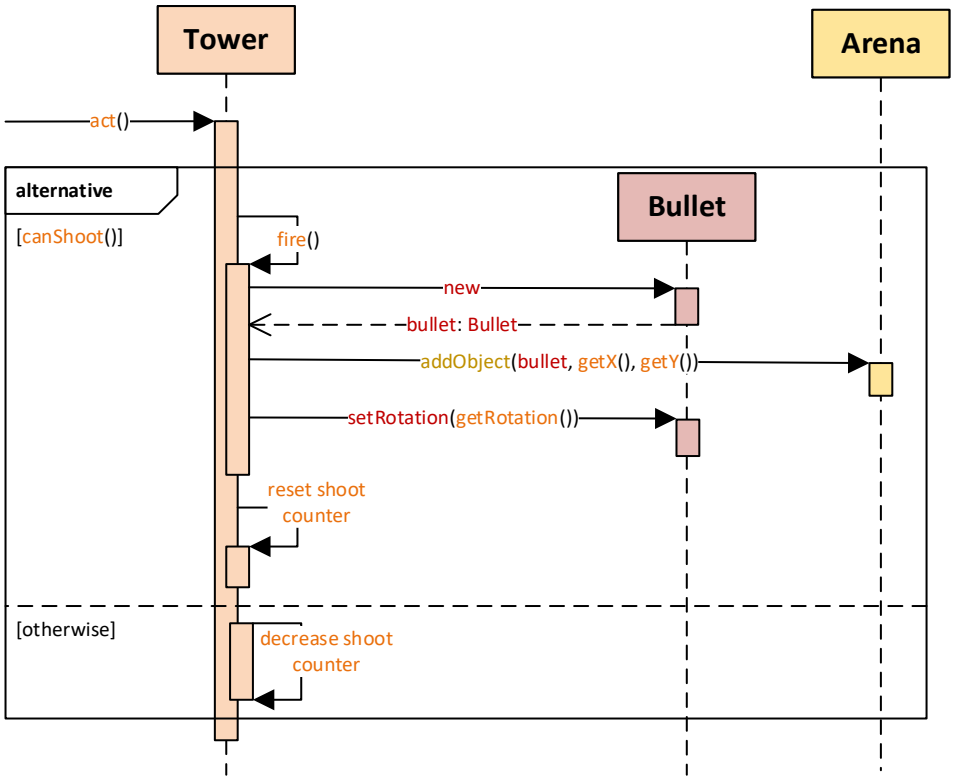


Abbildung 8: UML-Sequenzdiagramm der Instanz der Klasse Tower, die mit anderen Objekten interagiert, wenn Instanzen der Klasse Bullet erstellt werden

5.14. Implementieren Sie die Aufnahme einer Instanz der Klasse Tower

Verfolgen Sie die Ausgabe der Aufgabe 5.13:

- Bereiten Sie zuerst die notwendigen Attribute und den Konstruktor vor, dann
- Erstellen Sie die **`booleschen Methoden Tower.canShoot()`** und **`void Tower.fire()`** (die erste lässt **`false zurück`**, die zweite lässt nichts tun, damit Sie sie in **`der Methode act())`** verwenden können) und dann
- Implementieren Sie den Körper der Methode **`act().`**

Implementieren Sie die Methode **`canShoot(),`** um **`true zurückzugeben`** , wenn der Shoot-Zähler 0 erreicht.

Implementieren Sie die Methode **`fire()`** wie folgt:

- Aufruf des Konstruktors der Klasse **`Bullet`** und Speicherung der erstellten Instanz in der lokalen Variablen (**`Bullet Bullet`**),
- Füge eine erstellte **`Kugel`** in die Arena mit den gleichen Koordinaten wie die Instanz der **`Klasse Turm`** hinzu (**`diese`**) und
- Stellen Sie **`die gleiche Drehung wie der`** Schießturm ein.

Testen Sie Ihre Lösung.

5.15. Türme in der Arena

Überladungskonstruktor der Klasse **`Turm`** also akzeptiert es auch Parameter **`int Drehung`** (analog zu 4.8). Aktualisieren des Konstruktors der Klasse **`Arena`** So platzieren Sie Instanzen der Klasse **`Turm`** nach Wunsch. Verwenden des richtigen Konstruktors der Klasse **`Turm`**.

5.16. *Diskutieren Sie, wie die Instanz der Klasse Bullet mithilfe von Nachrichten mit den relevanten Objekten interagieren sollte*

Verwenden Sie analoge Prinzipien wie in 5.2.

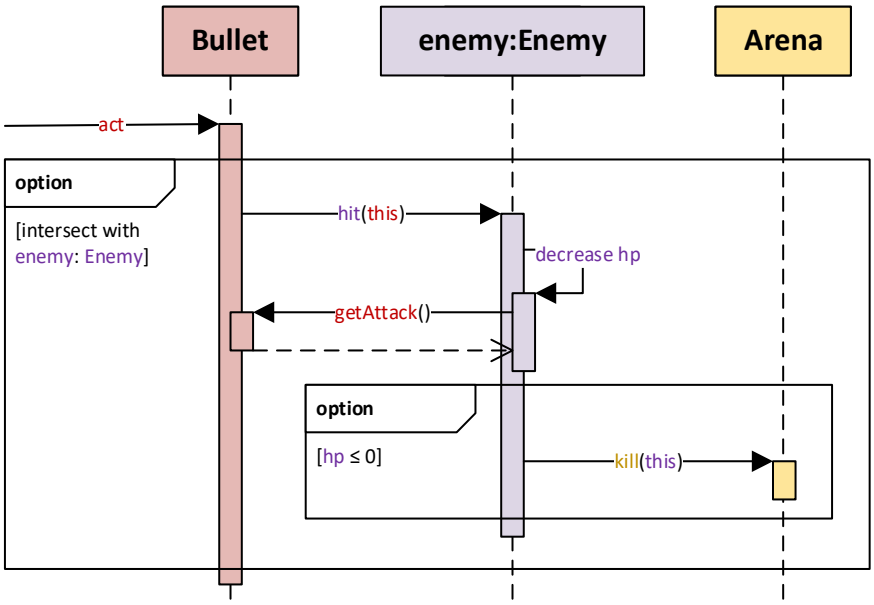


Abbildung 9: UML-Sequenzdiagramm einer Instanz der Klasse Bullet, die mit anderen Objekten interagiert, wenn sie auf eine Instanz der Klasse Enemy trifft

5.17. *Implementierung der Instanz der Klasse Bullet hitting der Klasse Enemy*

Verfolgen Sie die Ausgabe der Aufgabe 5.17:

- Vorbereiten von Attributen und Methoden (analog zu Aufgaben 5.3, 5.4, 5.5 und 5.6),
- Rufen Sie dann die Methode auf **`Enemy.hit(Kugel)`** von Instanz der Klasse **`Kugel`** (analog zu 5.7), von wo aus der Kommentar hinterlassen wurde 5.11 und
- Implementieren Sie schließlich die Methode **`Enemy.hit(Kugel)`** (analog zu 5.8).

Testen Sie Ihre Lösung.

5.18. *Spawn von Feinden und Ende des Spiels*

Verwenden Sie die Methode **`Arena.act(),`** um das Spawnen von Gegnern aufzurufen. Die Verzögerung zwischen dem Spawnen von Feinden wurde korrekt implementiert. Der Spawn-Prozess (Instanz der **`Klasse Enemy erstellen`**, ihre Eigenschaften zuweisen, sie in die Arena einfügen) wird in der **`Methode Arena.spawn()`** implementiert . Anzahl der erstellten Instanzen der **`Klasse Feind`** im Attribut der Klasse Arena (initialisiert auf 0, erhöht sich beim Spawnen, verringert sich bei Tötung). Alter-Methode **`Arena.kill(Enemy)`** – wenn der letzte Feind getötet wurde, hat der Spieler das Spiel gewonnen – stoppe Greenfoot und schreibe eine Nachricht auf den Bildschirm.

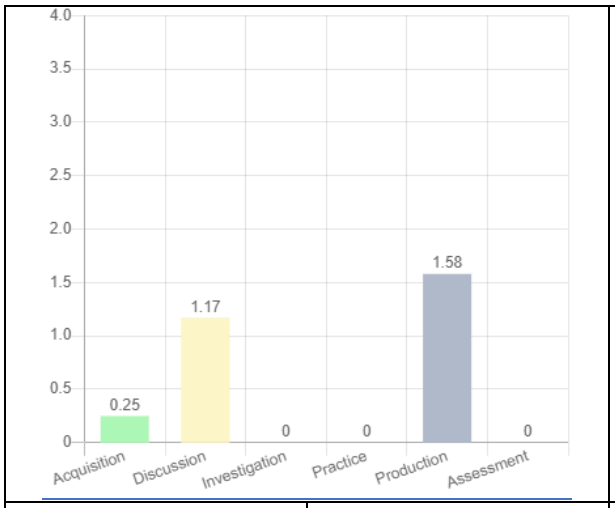
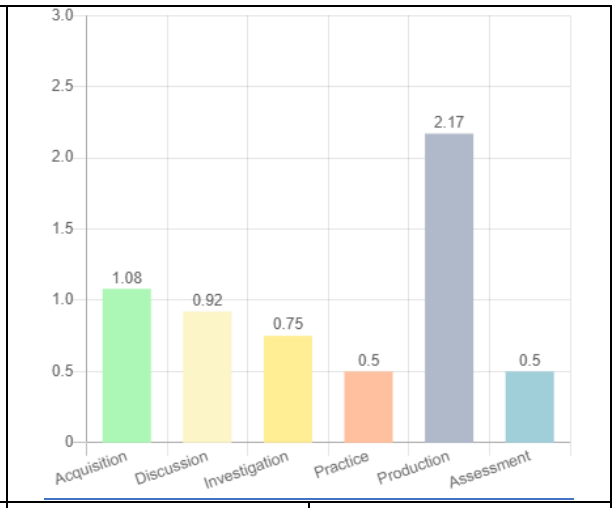
6. Vererbung

In diesem Thema wird die Variabilität über die Vererbung in das Projekt eingeführt. Wir führen das Liskov-Substitutionsprinzip ein, um die Vorteile von OOP aufzuzeigen. Wir empfehlen dringend, die Schüler experimentieren zu lassen und sich benutzerdefinierte Unterklassen von Gegnern und Arenen auszudenken. Da diese eine gemeinsame Schnittstelle haben, wird es einfach sein, alles zusammenzustellen. Ähnlich zum Thema 4 Es können viele Hausaufgaben erstellt werden.

Wie bereits erwähnt, kann das Projekt nach dem vorherigen Thema als abgeschlossen angesehen werden. Daher kann der Lehrer bei Bedarf dieses Thema anpassen, um die Vorteile der Vererbung und der damit verbundenen Universalität nur auf den hier vorgeschlagenen Hierarchien der Klasse **`Arena`** oder **`Feind zu zeigen`**. Dies führt zu einer Reduzierung der mit diesem Thema verbundenen Stunden.

Tabelle 9 fasst den Vergleich der Arbeitslasten des Themas Vererbung zwischen den Projekten Bomberman und Tower Defense zusammen. Der Vorschlag aus dem Experimentieren wird in mehr Untersuchungs-, Praxis- und Produktionstyp-TLAs projiziert. In diesem Thema wird mehr Theorie behandelt, wobei der Schwerpunkt auf dem Liskov-Substitutionsprinzip liegt.

Tabelle 9: Vergleich der Arbeitslasten des Themas Vererbung zwischen den Projekten Bomberman und Tower Defense

	Bomberman	3h 0min		Turmverteidigung	5h 55min

6.1. Gemeinsame Eigenschaften der Klassen Orb und Direction identifizieren

*Instanzen der Klassen Orb und Direction wirken während der Lebensdauer nicht. Sie reagieren nur auf Nachrichten. Wir können einen gemeinsamen Vorfahren einführen, der die Methode **`act()`** leer lässt und die Unterklassen transparenter macht.*

6.2. Hinzufügen der Klasse PassiveActor als Vorgänger der Klassen Orb und Direction

Erstellen Sie eine neue Klasse **`PassiveActor`**. Ändern Sie die Codes der Klassen **`Orb`** und **`Direction`** so, dass sie Nachkommen von **`PassiveActor sind`**. Entfernen Sie die Methode **`act()`** aus den Klassen **`Orb`** und **`Direction`** – sie wird von **`PassiveActor geerbt`**.

6.3. Klasse PassiveActor abstrakt machen

Beim Erstellen von Methoden der **`PassiveActor-Klasse`** stoßen wir auf Methoden, die in der allgemeinen Klasse nicht implementiert werden können, so dass wir die Implementierung den Nachkommen überlassen müssen. Betrachten wir als Beispiel die Shape-Klasse mit den untergeordneten Elementen **`Rectangle`** und **`Triangle`**. Für jede Form sind Perimeter- und Inhaltsmethoden implementiert, die jedoch nicht in einer gemeinsamen Klasse implementiert werden können. Wenn wir eine Klassenmethode als abstrakt markieren, sagen wir im Wesentlichen, dass ein Nachkomme sie implementieren wird. Die Klasse, die die abstract-Methode enthält, muss abstract sein. Daher fügen wir dem Klassenheader das Wort **`abstract`** hinzu.

6.4. Gemeinsame Eigenschaften der Klassen Bullet und Enemy zu identifizieren

Instanzen der Klassen Bullet und Enemy verhalten sich während der Lebenszeit ähnlich. Sie bewegen sich auf die gleiche Weise und reagieren danach auf die Umgebung. Wir können einen gemeinsamen

Vorfahren einführen, der die Methode ***act() implementiert,*** *um sich auf die gleiche Weise zu bewegen und Unterklassen auf ihren spezifischen Zweck zu konzentrieren.*

6.5. Fügen Sie die abstrakte Klasse MovingActor als Vorfahren der Klassen Bullet und Enemy hinzu

Verwenden Sie einen ähnlichen Ansatz wie in 6.2.

6.6. Attribute der Klassen Bullet und Enemy identifizieren, die für die Bewegung erforderlich sind

Untersuchen Sie die Methode **act()** der jeweiligen Klassen. Identifizieren Sie die Attribute **moveDelay** und **nextMoveCounter**. Beachten Sie, dass der Code der Methode **act(), der** für die Bewegung verantwortlich ist, derselbe ist.

6.7. Verschieben von Code, der für die Verschiebung verantwortlich ist, in die Klasse MovingActor

- Verschieben von Attributen, die in 6.6 aus Unterklassen **Kugel** und **Feind** in **Bewegender Akteur** (entfernen Sie sie aus Unterklassen).
- Fügen Sie der Klasse MovingActor **einen parametrischen Konstruktor hinzu** , um diese Attribute zu initialisieren.
- Rufen Sie den übergeordneten Konstruktor mit den richtigen Parametern aus den Unterklassen **Bullet** und **Enemy auf**.
- Verschieben Sie den Code, der für die Bewegung in der Methode **act()** der Unterklassen **Bullet** und **Enemy** verantwortlich ist, in **MovingActor** (entfernen Sie den Code aus den Unterklassen, behalten Sie dort den Rest der Methode).
- Rufen Sie die Elternversion der Methode **act()** als erste Zeile der Methode **act()** in den Unterklassen **Bullet** und **Enemy auf**.

6.8. Erstelle benutzerdefinierte Feinde

- Füge Unterklassen der Klasse **Feind hinzu** , die verschiedene Feinde repräsentieren (z.B. **Frosch** und **Spinne**). Stellen Sie sicher, dass die Bilder die Größe der Zelle nicht überschreiten.
- Fügen Sie den Klassen einen parameterlosen Konstruktor hinzu, der **den übergeordneten Konstruktor (Enemy) mit Parametern aufruft, die für jede Art von Feind spezifisch sind.**
- Entfernen Sie die Methode **act()** (oder fügen Sie den Aufruf **super hinzu**).

6.9. Spawne benutzerdefinierte Feinde

Aktualisieren Sie die Methode **Arena.spawn().** Erstellen Sie eine Instanz von **Frog** oder **Spider** und speichern Sie sie in der Variablen vom Typ **Enemy**. Verwenden Sie eine beliebige Art von Entscheidung, um zu entscheiden, welche Instanz erstellt werden soll (kann zufällig sein, kann genau gezählt werden usw.). Achten Sie darauf, dass keine anderen Codes in der Anwendung geändert werden müssen.

6.10. Diskutieren Sie die Hierarchie der Arenen

Diskutieren und entwerfen Sie die Hierarchie von **Arena-Klassen**. Die Unterklassen der **Arena** sind für das benutzerdefinierte Layout verantwortlich – Position der **Kugel**- und Richtungsinstanzen, Größe der Arena. Diese Aufgaben werden im Konstruktor der Unterklasse ausgeführt. Was wird benötigt, um Konstruktorparameter an die übergeordnete Klasse (**Arena**) zu übergeben? Denkt daran, dass der Rest (Spawnen, Respawnen und Töten von Gegnern) in der Klassenarena ausgeführt wird.

Sie sollten die Notwendigkeit erkennen, die Laichposition und -drehung einzustellen und zu speichern. Dies geschieht unter Verwendung von Attributen und relevanten Konstruktorparametern. Darüber hinaus sollte der Konstrukteur auch die Abmessungen der Fläche akzeptieren.

6.11. Universelle Arena herstellen

Führen Sie die Attribute **int spawnPositionX** , **int spawnPositionY** und **int spawnRotation ein** und verwenden Sie sie in den Methoden **spawn()** und **respawn(Enemy).** Fügen Sie dem Konstruktor der Klasse **Arena Parameter** hinzu, um sie zu initialisieren.

Fügen Sie dem Konstruktor der Klasse Arena zwei weitere Parameter hinzu – **int width** und **int height**. Übergeben Sie diese Parameter an den übergeordneten Konstruktor.

Beachten Sie, dass **Arena** nicht automatisch von **Greenfoot erstellt werden kann**, da es Parameter für den Konstruktor benötigt. Machen Sie es **abstrakt**.

6.12. DemoArena erstellen

Unterklasse **DemoArena** der Klasse **Arena hinzugefügt**. Rufen Sie den übergeordneten Konstruktor der Klasse **DemoArena** mit Parametern auf, die sicherstellen, dass eine Arena mit den gleichen Abmessungen und spawnenden Gegnern auf die gleiche Weise wie zuvor erstellt wird.

Verschieben Sie den Code, der für das Layout der Instanzen der Klassen **Direction**, **Orb** und **Tower verantwortlich ist,** vom Konstruktor der **Arena** zum Konstruktor der **DemoArena**.

Instanz der **DemoArena erstellen** – aus dem Kontextmenü der **Klasse DemoArena** den Punkt **new DemoArena() auswählen**.

6.13. Erstellen Sie benutzerdefinierte Arenen

Verwendung eines ähnlichen Ansatzes wie in 6.12 Erstellen Sie weitere innovative Unterklassen der Klasse **Arena**. Sie können den Code mit anderen Teilnehmern in Ihrer Gruppe teilen.

7. Verkapselung

Das letzte Thema befasst sich mit der korrekten Nutzung von privaten Methoden und Klassenmethoden und -variablen. Die Verwendung von Klassenmethoden und Variablen kann durch (Nicht-Klassen-)Attribute und Methoden in der Klasse Arena ersetzt werden (im Rahmen unseres Projekts gibt es nur eine Instanz von **Arena**). Dies wird es ermöglichen, Aufgaben aus diesem Thema umzusetzen, jedoch unter Verwendung der bereits bekannten Konzepte.

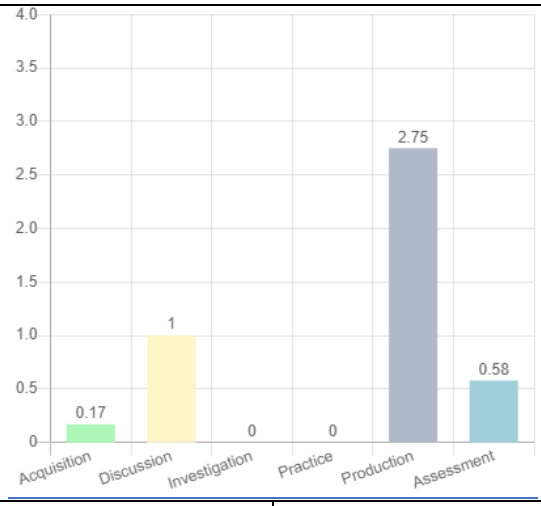
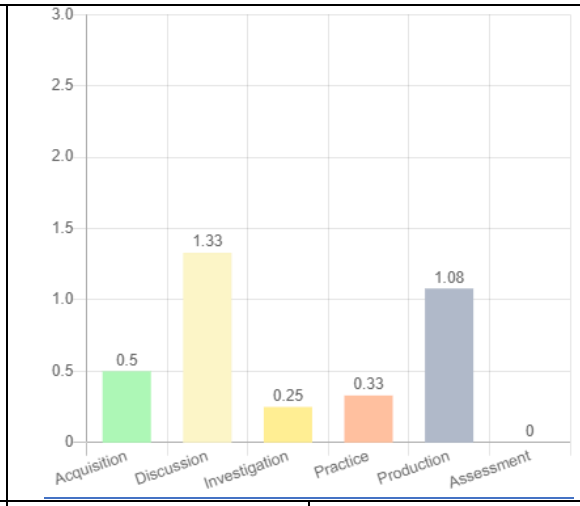
Wir haben die Kapselung bereits für das Bomberman-Projekt verwendet. Hier kommen statische Attribute und statische Methoden zum Einsatz. Bisher waren alle Attribute und Methoden an eine Instanz der Klasse gebunden. Stellen wir uns eine Situation vor, in der wir die in der Bullet-Klasse abgefeuerten Kugeln zählen möchten . Die Anzahl der abgefeuerten Kugeln ist für alle Instanzen der **Bullet-Klasse** gleich und hängt nicht von einer bestimmten Instanz ab. Ein solches Attribut wird als statisch bezeichnet und da es der Klasse gemeinsam ist, wird über den Klassennamen, z. B. **Bullet.count, darauf zugegriffen**. Auf ähnliche Weise können wir zum Beispiel eine Methode erstellen, die die Anzahl der Aufzählungszeichen zurückgibt. Auch dies gilt für alle Instanzen der Klasse. Es wird daher statisch sein. Statische Attribute und statische Methoden haben das Schlüsselwort **static** in ihrer Deklaration. Beachten Sie, dass sie auch dann vorhanden sein können, wenn keine Instanz der Klasse vorhanden ist. Statische Attribute werden in der Definition selbst initialisiert.

```
static int countOfBullets = 0;

static int countOfBullets(){

...
```

Tabelle 10 fasst den Vergleich der Arbeitslasten des Themas Kapselung zwischen den Projekten Bomberman und Tower Defense zusammen. Ähnlich wie im vorherigen Thema ist die Produktion gleichmäßiger auf andere Arten von TLAs verteilt. Die höhere Anzahl von Akquisitions-TLAs ergibt sich

aus der Einführung von Klassenmethoden und Variablen. Wie vorgeschlagen, gibt es eine Möglichkeit, diese Konzepte zu vermeiden, was zu einer Reduzierung dieser TLAs führen wird, was zu einem ähnlichen Design wie beim Projekt Bomberman führen wird.

Tabelle 10: Vergleich der Arbeitslasten des Themas Kapselung zwischen den Projekten Bomberman und Tower Defense

	Bomberman	4h 30min	Turmverteidigung	3h 30min
				

7.1. Klasse erstellen ManualTower

Erstellen Sie die **`Klasse ManualTower`** als Nachkomme der **`Klasse Tower`**. Bereiten Sie Bilder dieser Klasse vor, wenn sie unter Kontrolle sind und wenn nicht. Fügen Sie zwei Konstruktoren mit derselben Signatur wie die übergeordneten Konstruktoren hinzu, und stellen Sie sicher, dass der übergeordnete Konstruktor aufgerufen wird. Lassen Sie die Methode **`act()`** die übergeordnete Version von sich selbst aufrufen.

Füge Instanzen dieser Klasse in das Layout der gewählten **`Arena ein`**.

7.2. Änderung der Steuerung des manuell gesteuerten Turms

Fügen Sie das boolesche Attribut **`isManuallyControlled hinzu`** , und initialisieren Sie es mit **`false`**. Erstellen Sie die Methode **`void changeControl(boolean)`** und ändern Sie das Attribut und das Bild entsprechend.

7.3. Änderung der manuellen Steuerung aufrufen

Manuelles Aufrufen der Änderung der manuellen Steuerung der ausgewählten Instanz von **`Klasse ManualTower`**. Beobachten Sie Änderungen des internen Zustands ähnlich wie 1.6.

7.4. Benutzersteuerung für den Prozess

Erstellen Sie die private Methode **`void processUserControl().`** Erkennen Sie zunächst, ob es auf diese Instanz geklickt wurde. Wenn ja, wechseln Sie zur manuellen Steuerung. Anschließend implementieren Sie die manuelle Steuerung selbst. Testen Sie, ob sich die Instanz im manuellen Modus befindet, und wenn ja, rufen Sie **`das MouseInfo-Objekt`** ab . Wenn das Objekt abgerufen wurde, drehen Sie die Instanz von **`ManualTower`** in Richtung der Position des Mauszeigers.

Rufen Sie die vorbereitete Methode **`processUserControl()`** aus der Methode **`act() auf`** , bevor Sie die Ausführung an das übergeordnete Element übergeben (**`super.act()).`** Prüfen Sie in der Methode, ob sie auf diese Instanz geklickt wurde. Wenn dies der Fall ist, rufen Sie die Methode **`changeControl`** mit dem richtigen Wert auf.

Testen Sie Ihre Lösung, indem Sie 7.3 erneut ausführen.

7.5. Identifizieren Sie das Problem mit der Benutzersteuerung und schlagen Sie eine Lösung vor

Identifizieren Sie, was bei der Benutzersteuerung problematisch ist. Wie können diese Probleme gelöst werden?

Derzeit ist es nicht möglich, die Abwahl des Turms aufzuheben. Es sollte einen Nachweis für die aktuell kontrollierte Instanz vorhanden sein, der deaktiviert wird, wenn eine andere Instanz ausgewählt wird. Fügen Sie den Nachweis eines manuell gesteuerten Turms hinzu.

7.6. Hinzufügen von Nachweisen für manuell gesteuerten Turm

Fügen Sie der Klasse **`ManualTower`** vom Typ **`ManualTower ein Attribut`** hinzu, um den Verweis auf eine manuell gesteuerte Instanz darzustellen, und initialisieren Sie sie mit **`null`**. Dieses Attribut muss für alle Instanzen der ManualTower-Klasse gleich sein und daher mit dem Schlüsselwort **`static definiert werden`** . Überprüfen Sie den internen Zustand der Klasse (wählen Sie aus dem Kontextmenü der **`Klasse ManualTower`** den Menüpunkt **`Inspect`**). Was wurde hinzugefügt?

7.7. Wechsel des manuell gesteuerten Turms von einem zentralen Ort

Fügen Sie **`die Methode changeControlledInstance`** hinzu, um den manuell gesteuerten Turm als Klassenmethode der Klasse **`ManualTower`** zu ändern (also mit dem Schlüsselwort **`static definiert`**). Der Parameter der Methode sollte auf die Instanz von **`ManualTower`** verweisen , die manuell gesteuert wird.

Wenn sich der Methodenparameter von der manuell gesteuerten Instanz (in einem statischen Attribut) unterscheidet, verwenden Sie die **`changeControll-Methode`** mit den richtigen Parametern. Verwenden Sie **`zunächst changeControll`**, um die ursprüngliche manuell gesteuerte Instanz auf **`false festzulegen`**. Ändern Sie dann das statische Attribut der ManualTower-Klasse der manuell gesteuerten Instanz in den Parameter der neuen Methode (d. h. des neuen manuell gesteuerten Turms). Verwenden Sie abschließend die **`changeControll-Methode`**, um die neue manuell gesteuerte Instanz auf **`true`** festzulegen. Vergessen Sie nicht, mögliche NULL-Referenzen zu behandeln. Dies liegt daran, dass es eine Situation geben kann, in der es keinen manuell gesteuerten Turm gibt.

Testen Sie Ihre Lösung. Aus dem Kontextmenü von **`Klasse Tower`** Wählen Sie den Menüpunkt mit neu erstellter Methode. Um einen Parameter zu füllen, können Sie das gleiche Prinzip wie in 5.5.

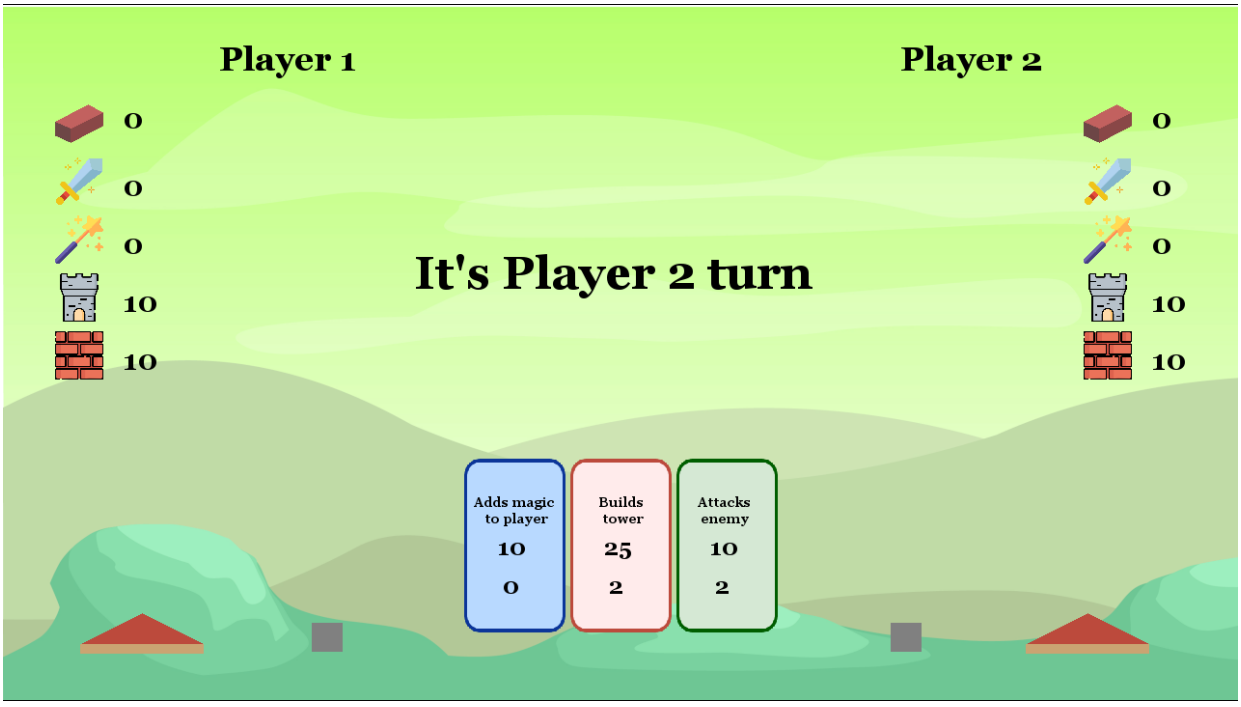
*Sie sollten beachten, dass die neu erstellte Klassenmethode nicht konsistent aufgerufen wird, Instanzen von **`ManualTower`** umgehen bei der Verarbeitung von Eingaben den Beweis, was Probleme verursacht.*

7.8. Wechsel des manuell gesteuerten Turms aufrufen

Anrufen **`ManualTower.changeControlledInstance(ManualTower)`** von relevanten Orten. Zuletzt Methode machen **`ManualTower.changeControl(boolescher Wert)`** privat. Beobachten Sie Änderungen in der Schnittstelle der Instanz von **`Klasse ManualTower`** Ähnlich wie bei 1.6.

3.3. Projekt Ameisen (Ameisen)

Ants ist ein kartenbasiertes Spiel für zwei Spieler. Jeder Spieler hat seinen eigenen Turm und seine eigene Mauer und Ressourcen wie Ziegel, Schwerter und Magie. Ein Spielzug besteht aus einer Aktion, bei der das Spiel dem Spieler 3 Karten nach dem Zufallsprinzip anbietet und er eine auswählt. Es gibt drei Arten von Karten – Baukarten, Kampfkarten und magische Karten. Baukarten können verwendet werden, um den eigenen Turm oder die eigene Mauer zu vergrößern, Kampfkarten, um einen feindlichen Spieler anzugreifen, und magische Karten, um die Anzahl der eigenen Ressourcen zu erhöhen oder die des Feindes zu stehlen.



Der Quellcode wurde in GIT unter der folgenden URL bereitgestellt:
https://gitlab.kicon.fri.uniza.sk/oop4fun/project-ants

Das Lerndesign kann unter der folgenden URL abgerufen werden:
http://learning-design.eu/en/preview/67aa1d089763d07f29809d42/details

Themen

Das Projekt Ameisen gliedert sich in 6 Themenbereiche:

1. Greenfoot-Umgebung, Klassendefinition, grundlegende Arbeit mit Klassen
2. Verkapselung, Zusammensetzung, Methoden
3. Konstruktoren, komplexere Methodenaufrufe (Arbeiten mit Grafiken in Greenfoot)
4. Verzweigung, bedingte Ausführung
5. Algorithmus, Aufzählungen, Arrays
6. Umgang mit Benutzereingaben, Spiellogik

Die behandelten Themen von Light-OOP sind die folgenden:

- Klassen, Objekte, Instanz
- Methoden, Übergeben von Methodenargumenten
- Erbauer
- Attribute
- Statische Variablen und Methoden
- Verkapselung

1. Greenfoot-Umgebung, Klassendefinition, grundlegende Arbeit mit Klassen

Das Thema widmet sich der Projekterstellung. Die Schüler werden in der Lage sein, ein neues Projekt in der Greenfoot-Umgebung zu erstellen, eine Klasse (als Unterklasse von Actor) zu erstellen, ein Hintergrundbild auszuwählen, eine Instanz der Klasse zu erstellen und ihr eine Nachricht zu senden.

Erstellen Sie ein neues Projekt. Geben Sie ihm einen richtigen Namen (z. B. Ameisen) und speichern Sie ihn an einem geeigneten Ort.

Tabelle 11 fasst den Vergleich der Arbeitslasten des ersten Themas zwischen den Projekten Bomberman und Ants zusammen. Die Gesamtauslastung beider Projekte ist in diesem Teil gleich.

Tabelle 11: Vergleich der Arbeitsbelastungen von Thema 1 zwischen den Projekten Bomberman und Ants

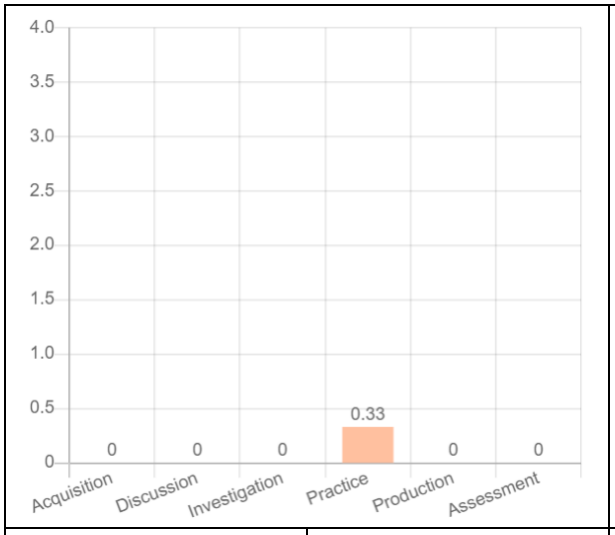
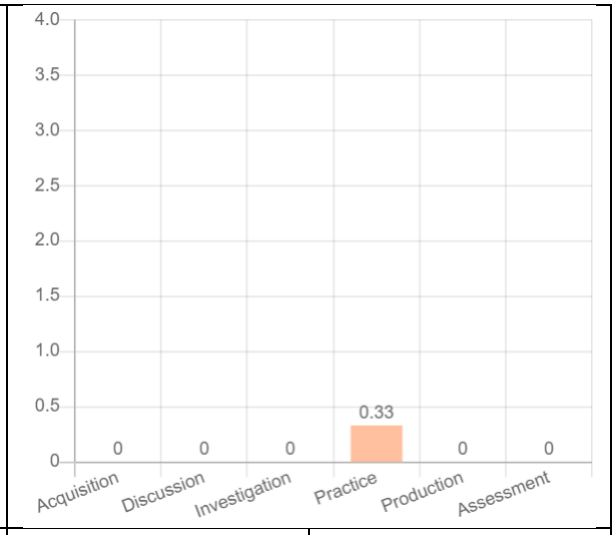
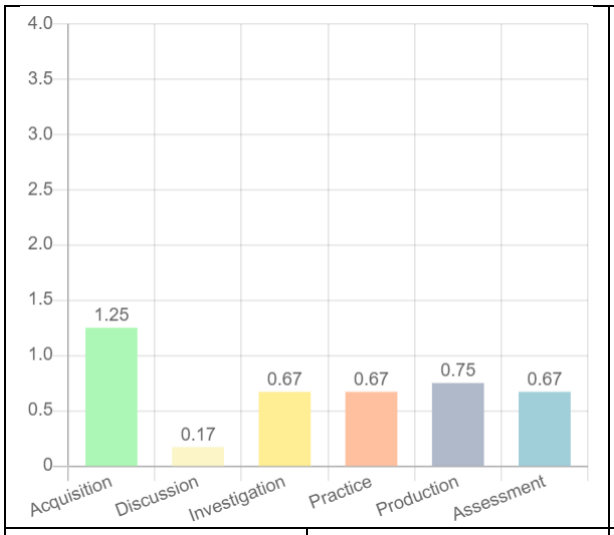
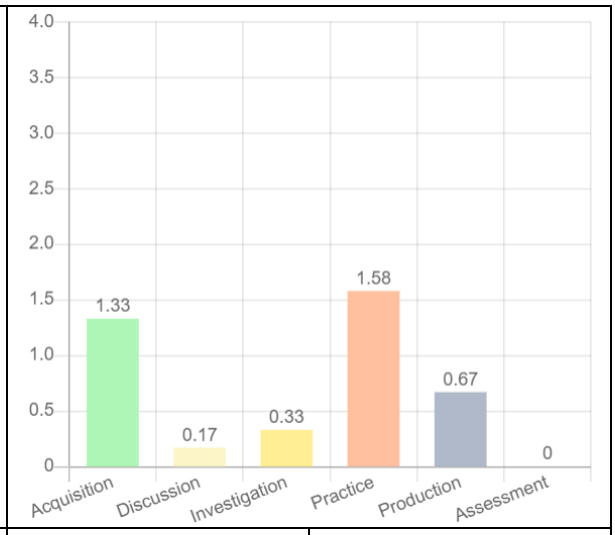
			
Bomberman	0h 20min	Ameisen	0h 20min

Tabelle 12 fasst den Vergleich der Arbeitslasten des Themas Klassendefinition und der grundlegenden Arbeit mit Klassen zwischen den Projekten Bomberman und Ants zusammen. Die Gesamtauslastung beider Projekte ist ähnlich. Der Hauptunterschied besteht in der Praxis und in der Bewertung. Wie jedoch im nächsten Abschnitt erläutert wird, können einige der Übungsaufgaben auch als Bewertungen gegeben werden, was das Ganze noch mehr ausgleichen kann.

Tabelle 12: Vergleich der Arbeitslasten des Themas Klassendefinition und Basisarbeit mit Klassen zwischen den Projekten Bomberman und Ants

	Bomberman	4h 10min		Ameisen	4h 05min

1.1. Einführung in Greenfoot

Erstellen Sie ein neues Projekt in Greenfoot und stellen Sie den Schülern grundlegende Elemente, Benutzeroberfläche usw. vor. Der Anfangsstatus des Repositorys enthält auch Assets, die Sie im Projekt verwenden können.

1.2. Erstellung der Klasse Wall

Erstellen Sie die Klasse Wall als untergeordnetes Element von Actor. Stellen Sie den Schülern Konzepte wie Klassen, Klassenhierarchie, Instanzen usw. vor.

1.3. Schaffung der Klasse Tower

Erstellen Sie auf ähnliche Weise wie bei der vorherigen Aufgabe die Klasse Tower. Sie können sie den Studierenden als Einzelaufgabe überlassen.

1.4. Definieren von Klassenattributen/-feldern

Stellen Sie den Schülern die Begriffe Feld/Attribut, primitive Typen usw. vor. Versuchen Sie, Felder in unseren Klassen zu identifizieren (führen Sie die Schüler speziell zur Höhe) und definieren Sie diese in der Klassenwand.

1.5. Zuweisen eines Werts zu einem Attribut/Feld

Sprechen Sie über Feldwerte und Zuweisungen, und weisen Sie sie der Wandhöhe den Wert 10 zu.

1.6. Definieren und Zuweisen eines Wertes zu einem Attribut/Feld für die Klasse Tower

Als individuelle Arbeit lassen Sie die Schüler das Gleiche für die Klasse Tower wiederholen.

1.7. Klassenkonstruktoren

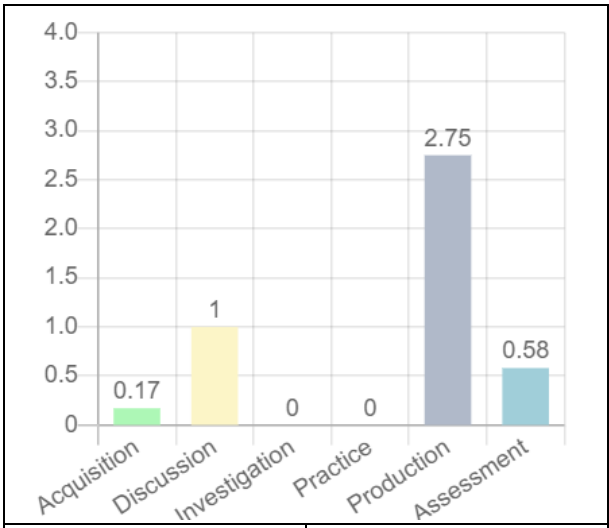
Sprechen Sie über die Instanziierung von Klassen und Konstruktoren. Verschieben Sie die Zuweisung des Werts zum Konstruktor.

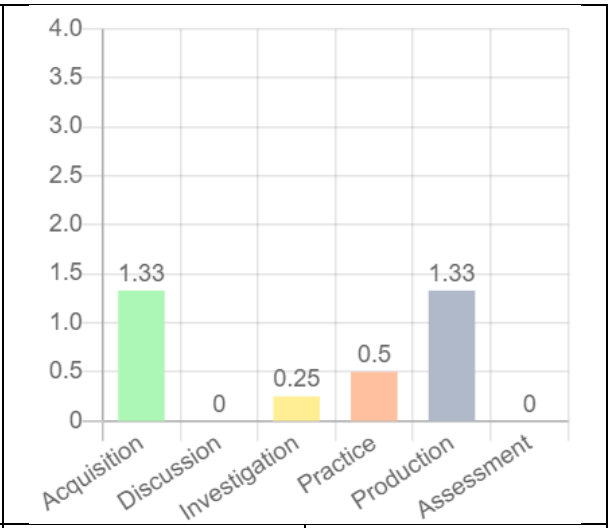
2. Verkapselung, Zusammensetzung, Methoden

In diesem Abschnitt werden die Grundprinzipien von OOP vorgestellt, insbesondere Begriffe wie Verkapselung, Zusammensetzung und Methoden. Die Studierenden lernen, wie und warum Felder/Attribute gekapselt und nicht öffentlich zur Verfügung gestellt werden sollten, wie Objekte mit anderen Objekten zusammengesetzt sind und wie Methoden erstellt und aufgerufen werden. In diesem Abschnitt erstellen die Schülerinnen und Schüler ein Objekt des Spielers und geben ihm Felder des Objekttyps – Mauer und Turm.

Tabelle 13 zeigt Unterschiede in zwei ähnlichen Themen – Verkapselung aus dem Bomberman-Projekt und Verkapselung, Zusammensetzung und Methoden in Ameisen. Wie man sehen kann, konzentriert sich das Ants-Projekt mehr auf die Akquisition und auf der anderen Seite konzentriert sich Bomberman mehr auf Produktion und Diskussion. Dies liegt daran, dass dieses Thema bei Ameisen aus mehr als nur Verkapselung besteht, daher ist auch mehr Akquisition erforderlich. Die Gesamtarbeitsbelastung ist in Ants eine Stunde kürzer als in Bomberman, da diese Konzepte oberflächlicher erklärt werden, aber in den nächsten Abschnitten würden Sie dieses Wissen festigen.

Tabelle 13: Vergleich der Arbeitslasten des Themas Kapselung, Komposition, Methoden zwischen den Projekten Ameisen und ähnlichem Thema in Bomberman - Kapselung





Bomberman	4h 30min	Ameisen	3h 25min

2.1. Definieren von Methoden

Sprechen Sie über Kapselung, erklären Sie den Schülern, warum es nicht gut ist, z.B. die Wandhöhe als öffentliches Eigentum zu machen und sie lieber im Getter einzukapseln. Erstellen Sie dann die Methode getHeight() in Wall.

2.2. Definieren von Methoden mit Parametern

Erläutern Sie die Methodenparameter und definieren Sie die Methode increaseHeight in wall, die die Wandhöhe um die angegebene Zahl erhöht.

2.3. Wiederholung für die Klasse Tower

Als Einzelarbeit können Sie den Schülern die Aufgabe geben, sie auch für den Klassenturm zu wiederholen. Dieser Aufgabe ist kein Commit zugeordnet, da die Arbeit recht einfach ist. Im nächsten Commit gibt es auch Änderungen für diese Aufgabe, sodass Sie sie vergleichen können, wenn Sie sich über das Ergebnis nicht sicher sind.

2.4. Zusammensetzung des Objekts

Erklären Sie den Schülern, was Komposition ist und warum es notwendig ist, Objekttypen als Felder zu haben. Versuchen Sie, solche Typen für jeden Spieler in diesem Spiel zu identifizieren. Erstellen Sie dann das Objekt Player und geben Sie ihm die Felder/Attribute Wall und Tower.

2.5. Erstellen einer Instanz einer Klasse

Erklären Sie Konstruktoren und erstellen Sie Wand- und Turminstanzen im Player-Konstruktor.

2.6. Aufrufen von Methoden der Instanz

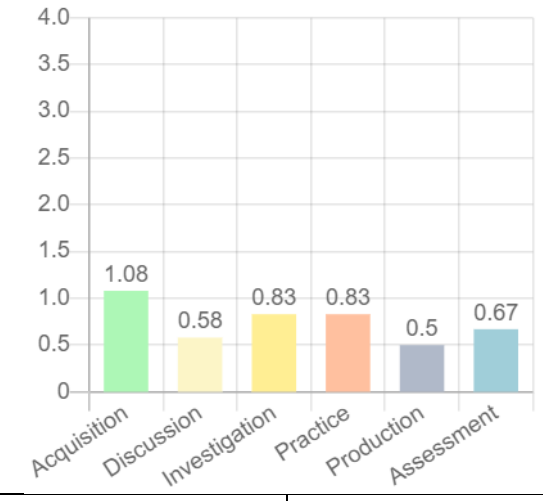
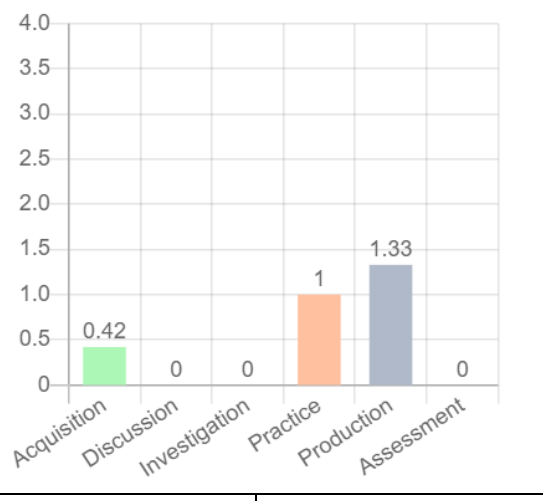
Erklären Sie den Teilnehmern, wie Sie Methoden von erstellten Instanzen aufrufen können. Versuchen Sie dann, sie so zu kapseln, dass man sie von außerhalb des Spielers über die Spielerinstanz aufrufen kann. Erstellen Sie die Methoden getWallHeight, getTowerHeight und increaseWallHeight, increaseTowerHeight im Player-Objekt.

3. Konstruktoren, komplexere Methodenaufrufe (Arbeiten mit Grafiken in Greenfoot)

Dieser Abschnitt konzentriert sich auf Methodenaufrufe, die den Aufruf von Greenfoot-Objekten zum Zeichnen unserer Instanzen verwenden. Die Schülerinnen und Schüler zeichnen Instanzen von Mauer, Turm und Spieler.

Tabelle 14 zeigt einen Vergleich dieses Themas im Projekt Ants mit dem ähnlichsten Thema im Projekt Bomberman. Bitte beachten Sie, dass diese Themen etwas unterschiedlich sind, da unseres die grundlegende Arbeit mit Konstruktoren zeigt und Grafiken in Greenfoot einführt und im Projekt Bomberman mehr auf Algorithmen ausgerichtet ist. Daher kann auch aufgrund dessen auf die Unterschiede hingewiesen werden.

Tabelle 14: Vergleich der Arbeitslasten des Themas Konstruktoren, komplexere Methodenaufrufe (Arbeiten mit Grafik in Greenfoot) zwischen den Projekten Ants und ähnlichem Thema im Projekt Bomberman, das im Thema Algorithmus behandelt wird

			
Bomberman	4h 30min	Ameisen	2h 45min

3.1. Zeichnen von Objekten in Greenfoot – Wall

Führen Sie die Schüler in Konstanten in Java ein – definieren Sie wallSizeX und wallSizeY als statische finale (das Schlüsselwort final stellt sicher, dass der Wert nicht geändert werden kann, d.h. dass es sich um eine Konstante handelt) Attribute (Konstanten) mit den Werten 32 und 3. Der Zugriff auf die Konstante erfolgt als statische Variable über den Klassennamen, z.B. Wall.wallSizeX. Implementieren Sie dann die Funktion Zeichnen in der Wand, wo ein neues Bild mit der angegebenen Größe erstellt und als Rechteck gefüllt wird.

3.2. Zeichnen von Objekten in Greenfoot – Tower

Das Gleiche wird für den Turm wiederholt, ist jedoch komplizierter, da der Turm auch aus einem Dach besteht, das als Polygon ausgeführt ist. Für ein Polygon ist ein Array von Punkten erforderlich. Wenn Sie möchten, können Sie den Teilnehmern eine kurze Erklärung geben, obwohl Arrays nicht Teil dieses Abschnitts sind. Sprechen Sie mit den Schülerinnen und Schülern über die Analogie der Begriffe Array und Liste.

3.3. Definieren anderer Eigenschaften des Players

Versuchen Sie, andere Eigenschaften des Spielers zu identifizieren – insbesondere die Anzahl der Steine, Schwerter und Magie sowie den Namen des Spielers. Implementieren Sie dann diese Eigenschaften im Player, und initialisieren Sie sie im Konstruktor.

3.4. Zeichnungs-Spieler

Die letzte Aufgabe in diesem Abschnitt ist das Zeichnen eines Spielers. Sie müssen die Symbole jeder Ressource und die Anzahl dieser Ressourcen sowie den Namen des Spielers zeichnen.

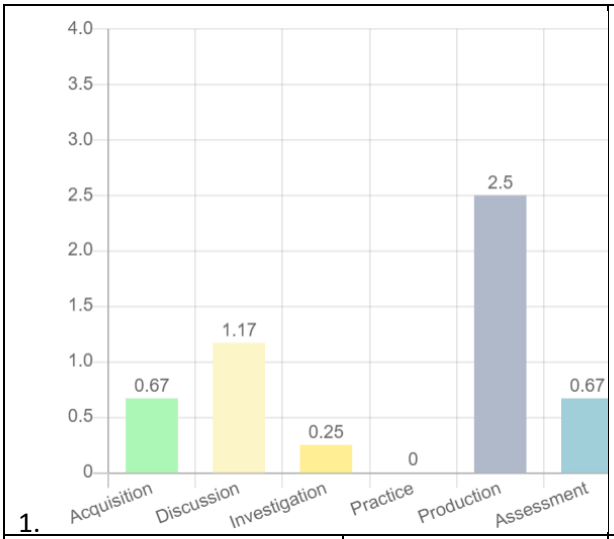
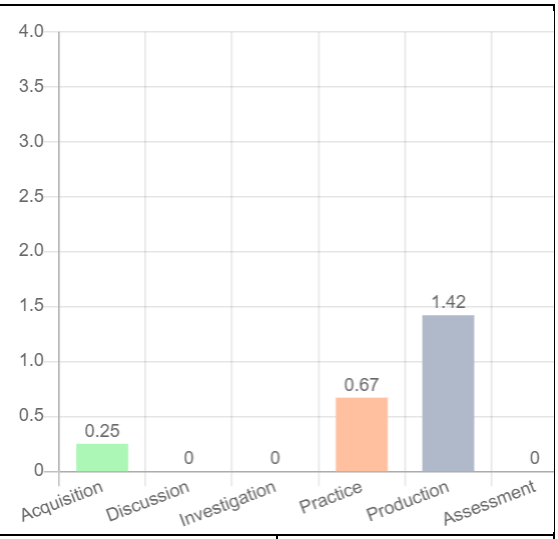
4. Verzweigung, bedingte Ausführung

Dieser Abschnitt konzentriert sich auf die Verzweigung von Programmen und die bedingte Ausführung von Teilen unseres Spiels. Es gibt mehrere Fälle, in denen dies in diesem Spiel notwendig ist, wie z.B. das Ziehen des ersten Spielers im linken Teil des Bildschirms und des zweiten im rechten Teil des Bildschirms usw.

Tabelle 15 zeigt Unterschiede zwischen ähnlichen Themen in Bomberman und Ants. Wie man sehen kann, legt das Bomberman-Projekt mehr Wert auf die Diskussion als das Projekt Ameisen. Außerdem ist die Gesamtarbeitsbelastung in Stunden etwa halb so hoch wie in Bomberman. Dies liegt daran, dass das Branching in diesem Projekt in mehrere Abschnitte aufgeteilt ist – dieser Abschnitt ist eher eine Einführung und grundlegende Verwendung des Branchings.

Tabelle 15 zeigt Unterschiede zwischen ähnlichen Themen in Bomberman und Ants. Wie man sehen kann, legt das Bomberman-Projekt mehr Wert auf die Diskussion als das Projekt Ameisen. Außerdem ist die Gesamtarbeitsbelastung in Stunden etwa halb so hoch wie in Bomberman. Dies liegt daran, dass das Branching in diesem Projekt in mehrere Abschnitte aufgeteilt ist – dieser Abschnitt ist eher eine Einführung und grundlegende Verwendung des Branchings.

Tabelle 15: Vergleich der Arbeitslasten des Themas Branching, bedingte Ausführung zwischen den Projekten Bomberman und Ants

1. 	3. Bomberman	4. 5h 15min	2. 	5. Ameisen	6. 2h 20min

4.1. Erstellen einer Klasse Spiel

Bevor wir anfangen, Verzweigungslogik in unseren Code zu integrieren, müssen wir ein Objekt Game erstellen, das beide Spieler hält und in Zukunft die Spiellogik, das Wechseln der Runden, die Kartenausführung usw. verwaltet. Im Moment werden wir die Eigenschaften für zwei Spieler einfügen und ihre Instanzen im Konstruktor der Game-Klasse erstellen.

4.2. Verzweigung, Konditionierung der Ausführung von Code – Spieler werden auf entsprechenden Seiten des Spielplans angezeigt

Jetzt sollten wir ein neues Attribut für unseren Player einführen – die Information, ob der Player auf der linken oder rechten Seite des Bildschirms dargestellt werden soll. Wir werden diese Informationen verwenden, um die entsprechenden Eigenschaften des Spielers festzulegen – Position der Wand, des Turms, des "Huds" und des Namens. Dieser Versatz sollte dann der Methode redraw hinzugefügt werden.

4.3. Hinzufügen von Instanzen zu world

Die nächste Aufgabe erstellt eine erste Ansicht der Player-Klasseninstanzen für das Spiel und instanziiert das Spiel in der Welt.

4.4. Hinzufügen von Instanzen zu Welt 2

Damit der Spieler im Spiel korrekt gezeichnet wird, müssen wir der Welt eine Mauer und einen Turm hinzufügen und die Funktion "Neuzeichnen" in der Aktmethode aufrufen. Hier können Sie die Ausführung der Spielschleife (Act-Methode) erklären.

4.5. Ausführungscode nur einmal konditionieren

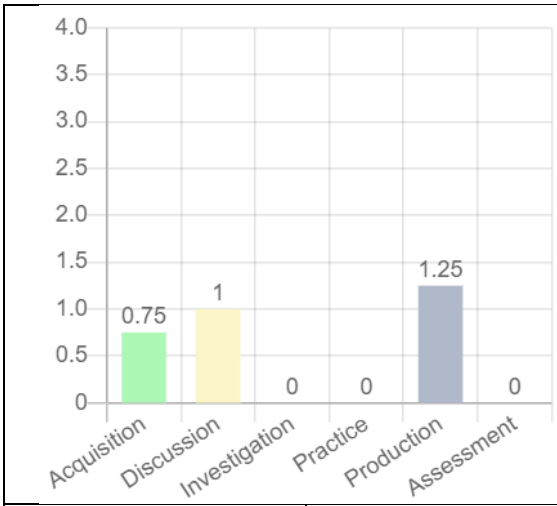
Wenn Sie versuchen, das Spiel nach der letzten Aufgabe auszuführen, kann es zu einem Problem kommen – Objekte werden der Welt mehrmals in einer Sekunde hinzugefügt. Sie können den Schülern Aufgaben geben, um dieses Problem zu beheben oder es mit ihnen zu beheben. Eine der Lösungen besteht darin, ein neues boolesches Attribut einzuführen, das Informationen speichert, ob das ursprüngliche Objekt, das der Welt hinzugefügt wurde, ausgeführt wurde oder nicht, und nach der ersten Ausführung der act-Methode wird diese Eigenschaft auf true gesetzt.

5. Algorithmus, Aufzählungen, Arrays

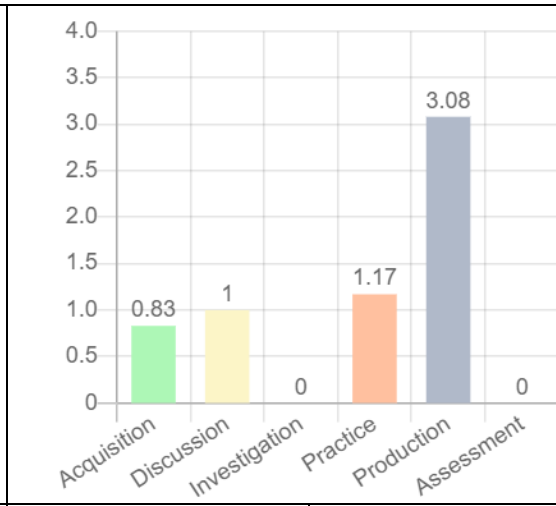
In diesem Abschnitt werden die nächsten Konzepte erläutert, z. B. Algorithmus, Enumerationen, Arrays und Schleife über Array von Elementen. In diesem Abschnitt werden die Schülerinnen und Schüler Karten implementieren, die zum Spielen des Spiels verwendet werden.

Tabelle 16 enthält einen Vergleich zwischen ähnlichen Themen in den Projekten Ants und Bomberman. Wie man sieht, legt das Projekt Ants einen stärkeren Fokus auf Produktion und Praxis und etwas mehr auf die Akquise. Außerdem ist die Gesamtauslastung etwa doppelt so groß wie im Projekt Bomberman. Dies liegt daran, dass dieses Thema nicht nur aus der Arbeit mit Listen besteht, sondern auch aus Konzepten wie Aufzählungen usw. besteht.

Tabelle 16: Vergleich der Arbeitslasten des Themas Algorithmus, Aufzählungen, Arrays im Projekt Ants und ähnliches Thema in Projekten Bomberman - Listen



Bomberman	3h 00min	Ameisen	6h 05min



5.1. Implementieren der Kartenklasse

Zuerst können wir mit den Schülern besprechen, wie das endgültige Spiel funktionieren würde, und das Kartenobjekt entwerfen. Sie sollten zu einer Lösung kommen, die Informationen über den Kartentyp, seine Anforderungen, seine Wirkung und eine Beschreibung enthält. Dann können Sie ein solches Objekt als untergeordnetes Element von Actor erstellen.

5.2. Aufzählungen

In der vorherigen Aufgabe haben Sie den Kartentyp als Feld im Kartenobjekt erstellt. Besprechen Sie mit den Schülern, welchen Typ dieses Feld haben sollte – String, int usw., und Sie können Enumerationen einführen. Enum ist ein Datentyp mit einer endlichen Menge von benannten Werten (z. B. für Wochentage sind es die Werte: Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag und Sonntag). Erstellen Sie mit ihnen eine CardType-Enumeration, und geben Sie die einzelnen Werte an.

5.3. Verzweigung – Schalter

Jetzt sollten wir das Zeichnen einer Karte implementieren. Dies basiert auf dem Kartentyp, um Karten visuell zu unterteilen. Sie können den Schülern zeigen, wie dies mit if gemacht wird, und es mit switch vergleichen. Es gibt drei Kartentypen – Baukarten, Angriffskarten und magische Karten. Basierend auf dieser Kategorie wird der Hintergrund ausgewählt. Falls wir dies mit der if-Anweisung tun, sollte der Code wie folgt aussehen:

```
if(type == BuildTower || type == BuildWall || type == IncreaseBricks) {
        Hintergrund = neu GreenfootImage("building-card.png");
} else if(type == IncreaseSwords || type == Angriff)
...
```

Dies kann durch die switch-Anweisung ersetzt werden. Die Java-switch-Anweisung funktioniert so, dass ein bestimmter Zweig ausgeführt wird, aber sie hört nicht auf, andere Zweige auszuführen, es sei denn, Sie rufen die break-Anweisung auf. In unserem Fall ermöglicht uns dies, mehrere Zweige zusammenzuführen und unsere Zuweisungsanweisung nur für den letzten Kartentyp dieser Kategorie zu schreiben. Daher führt dieser Code Folgendes aus:

```
switch (type) {
        case BuildTower:
                Hintergrund = neu GreenfootImage("building-card.png");
                break;
        case BuildWall:
                Hintergrund = neu GreenfootImage("building-card.png");
                break;
        case IncreaseBricks:
                Hintergrund = neu GreenfootImage("building-card.png");
                Break;
        ...
```

```
}
```

Mit einem switch-case-Block kann das auch so geschrieben werden:

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

5.4. Anordnung

Das Spielobjekt sollte Karten enthalten. Dies kann durch die Einführung von drei Feldern des Kartentyps erreicht werden (Sie können auch die Hand – d.h. die Anzahl der Karten, die das Spiel dem Spieler in einem Zug anbietet – auf mehrere von ihnen ausdehnen). Sie können erklären, dass es unmöglich wäre, noch mehr Karten zu speichern, wenn wir uns entscheiden, die Hand noch mehr zu verlängern, und Sie können den Schülern auch das Konzept der Arrays erklären. Sie sollten auch eine Instanz eines Arrays erstellen.

5.5. Vereinfachung der Instanziierung von Karten – CardFactory

Bei der Erstellung neuer Karten sollten viele Informationen bereitgestellt werden. Sie können versuchen, eine Lösung für dieses Problem zu finden. Eine der Lösungen besteht darin, eine CardFactory zu erstellen, die Instanzen für jede Karte enthält, und die Klonmethode auf der Karte zu implementieren. Daher können neue Karten mit dieser CardFactory erstellt und vorhandene geklont werden.

5.6. Zufällig – Instanziierung einer zufälligen Karte

Nach der Implementierung von CardFactory sollte auch die Klonmethode implementiert werden, wie zuvor besprochen wurde. Die CardFactory sollte auch in der Lage sein, Instanzen von zufälligen Karten auszugeben. Sie können den Zufallszahlengenerator erklären und eine Methode implementieren, die zufällige Karten und auch zufällige Basiskarten klont (Basiskarten sind Karten ohne Kosten - dies wird implementiert, damit wir garantieren können, dass der Spieler immer mindestens eine der bereitgestellten Karten spielen kann).

5.7. Schleife über Array

Jetzt müssen wir eine Instanz der CardFactory im Spiel erstellen und das Generieren von Karten und das Löschen (Entfernen aus der Welt) implementieren - wenn Sie dies erklären, können Sie eine Form der Schleife einführen.

5.8. Zeichnung des Wildes

Die letzte Aufgabe in diesem Abschnitt besteht darin, das Zeichnen des gesamten Spiels zu implementieren. Dabei sollten wir auch unsere Karten mit einer zuvor erstellten Methode vorbereiten und auch Informationen darüber einführen, ob der erste Spieler aktiv ist. Das Ziehen eines Spiels sollte

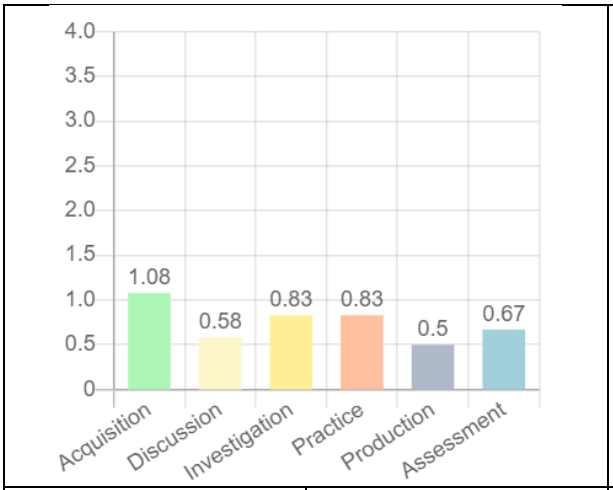
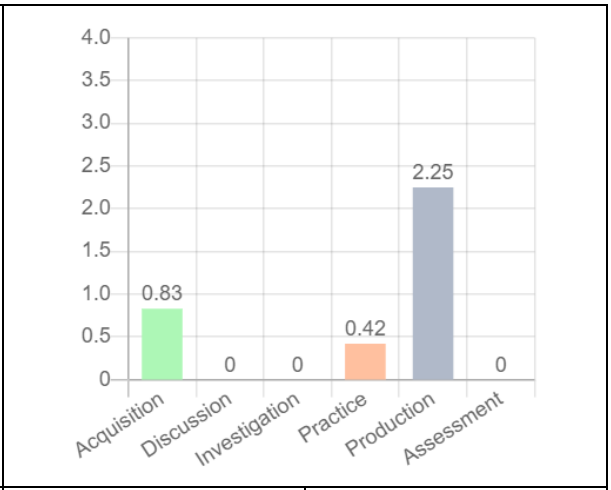
aus dem Ziehen aller Karten und der Ziehung von Informationen darüber bestehen, welcher Spieler gerade an der Reihe ist.

6. Umgang mit Benutzereingaben, Spiellogik

Dieser Abschnitt konzentriert sich auf den Umgang mit Benutzereingaben – wie man z.B. den Namen des Spielers vom Benutzer erhält, wie man mit dem Klicken auf Karten umgeht und wie man die Spiellogik beendet. In diesem Abschnitt werden auch einige fortgeschrittene Konzepte vorgestellt (Singleton).

Tabelle 17 zeigt einen Vergleich zwischen diesem Thema und dem ähnlichsten Thema im Projekt Bomberman – Algorithmus. Bitte beachten Sie, dass, da dies das letzte Thema dieses Projekts ist, viele Konzepte, die im Projekt Bomberman vorgestellt wurden, den Studierenden zu diesem Zeitpunkt dieses Projekts bereits bekannt sind. Daher wird deutlich weniger auf Recherche und Diskussion geachtet als im Projekt Bomberman und viel mehr produziert.

Tabelle 17: Vergleich der Arbeitslasten des Themas Umgang mit Benutzereingaben, Spiellogik im Projekt Ameisen und ähnlichem Thema im Projekt Bomberman - Algorithmus

			
Bomberman	4h 10min	Ameisen	3h 30min

6.1. Geben Sie Namen vom Benutzer ein

In dieser Aufgabe sollten Sie den Schülern Dialogfenster (Greenfoot-ask-Methode) erklären und die Spielernamen entsprechend diesen Eingaben festlegen.

6.2. Statische Instanz der Klasse – Spiel als Singleton

Spielobjekte sollten genau einmal konstruiert werden – Sie können dieses Problem mit den Schülern diskutieren und eine Lösung in Form eines Singletons anbieten – statische Instanz des Spielobjekts und des privaten Konstruktors. Singelton wird in Fällen verwendet, in denen nur eine Instanz einer bestimmten Klasse in der gesamten Anwendung vorhanden sein soll. Ein bekanntes Beispiel für ein Singleton ist z.B. das Package im Betriebssystem. Dies ist erforderlich, da wenn ein Spieler eine Karte verwendet, ein Bezug zum Spiel vorhanden sein sollte, damit das Anklicken korrekt gehandhabt werden kann, wie im nächsten Teil besprochen wird. Die andere Möglichkeit, es zu implementieren (ohne Singleton), besteht darin, dem Konstruktor der Karte und der CardFactory eine Instanz von Game zur Verfügung zu stellen.

6.3. Eingaben auf der Karte bearbeiten

Der Mausklick wird durch Aufrufen der Greenfoot.mouseClicked-Methode in act behandelt. Wenn Sie darauf klicken, sollten Sie die Methode useCard of Game aufrufen und den Verweis an sich selbst senden (this).

6.4. Implementierung von Gettern von Karte und Spieler

Bevor wir die Restlogik für das Spiel implementieren, benötigen wir noch einige Getter und Setter (Setter ist eine Methode zum Festlegen des Wertes privater Attribute) für Spieler und Karten. Also sollten wir sie implementieren (das sollte für die Schüler kein Problem sein, da es schon einmal gemacht wurde).

6.5. Implementierung der unterstützenden Methode für Player und Fix in World

Da das Spiel jetzt ein Singleton ist, müssen wir es der Welt in der MyWorld-Klasse hinzufügen. Die nächste Aufgabe besteht darin, eine unterstützende Methode für den Spieler zu implementieren, mit der er eine gewisse Menge an Schaden erleiden kann. Der Spieler sollte die Mauer oder den Turm entsprechend verringern.

6.6. Implementieren der Spiellogik

Schließlich müssen wir die Spiellogik – Turn-Methode implementieren, die aus den folgenden Schritten bestehen wird:

1. Prüfen, ob ein Spieler gewinnt – das heißt, wenn einer der Spieler die Höhe 100 erreicht oder unter den Wert 0 fällt. Wenn eine dieser Bedingungen erfüllt ist, zeigen wir den Gewinnbildschirm an und beenden die Spielschleife – Call Return-Anweisung.
2. Setzen Sie den aktiven Player auf einen anderen – drehen Sie einfach den Wert des isPlayer1Active-Attributs um.
3. Karten für den nächsten Spieler vorbereiten – da wir bereits die Methode prepareCards erstellt haben, müssen wir sie in diesem Schritt nur noch aufrufen.
4. Behandeln Sie den Zug der Spieler – insbesondere das Klicken auf Karten. Da die Karte selbst in der Lage ist, das Klicken darauf zu hören, müssen wir nur die Ziehmethode des Spiels aufrufen, bei der vorbereitete Karten gezogen werden.
5. Spieler neu zeichnen – dies geschieht mit der Methode Neuzeichnen für jeden Spieler.

Dann müssen wir die useCard-Methode in Game mit switch implementieren. Dieser Schalter sollte einen Zweig für jeden Kartentyp enthalten und seine Ausführung verarbeiten. Es gibt 7 Kartentypen: BuildTower, BuildWall, IncreaseBricks, IncreaseSwords, Attack, IncreaseMagic und StealBricks. Werfen wir einen Blick auf den ersten Kartentyp – BuildTower. In diesem Fall müssen wir prüfen, ob der Spieler diese Karte spielen kann (d.h. die Anzahl seiner Steine ist größer oder gleich den Kartenanforderungen), die Höhe des Trefferturms erhöhen und die Anzahl der getroffenen Steine auf die Kartenanforderungen verringern. Der Code kann also wie folgt aussehen:

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

{

activePlayer.increaseTowerHeight(card.getEffect());
```

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

Die anderen Kartentypen sind ähnlich:

- BuildWall – wir müssen nach der Anzahl der Steine der Spieler suchen und wir werden die Höhe der Mauer erhöhen.
- IncreaseBricks – wir müssen nicht nach irgendetwas suchen und erhöhen die BricksNumber
- IncreaseSwords – wir müssen nicht nach irgendetwas suchen und werden die Anzahl der Schwerter erhöhen
- Angriff – wir müssen nach den Schwertern der Spieler suchenNummer und Anruf erhaltenSchaden des inaktiven Spielers
- IncreaseMagic – wir müssen nicht nach irgendetwas suchen und werden magicNumber erhöhen
- StealBricks – wir müssen nach magicNumber suchen, die Anzahl der inaktiven Spielersteine verringern und die Anzahl der aktiven Spielersteine erhöhen.

Es ist auch möglich, andere Kartentypen zu erstellen – es ist für die Fantasie der Schüler. Dies sind einige grundlegende.

Schließlich müssen wir die Turn-Methode aufrufen, nachdem wir eine Karte gespielt haben, um die Spiellogik sicherzustellen.

Als letzten Schritt sollten wir einen Gewinnbildschirm implementieren. Dies ist wie bei anderen Zeichnungen in diesem Projekt, daher liegt es an Ihnen, es entweder mit den Schülern zu erstellen oder es ihnen zu überlassen.

Um das Ganze abzurunden, gibt es auch einige Korrekturen im Player und im Turm, um den Code zu bereinigen.

4. Bibliographie

[1] "Git", 1 10 2023. [Online]. Verfügbar: https://git-scm.com. [Zitat 1 10 2023].

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

[3] "GitHub: Lass uns von hier aus aufbauen · GitHub", 1 10 2023. [Online]. Verfügbar: https://github.com. [Zitat 1 10 2023].

[4] "Die DevSecOps-Plattform | GitLab", 1 10 2023. [Online]. Verfügbar: https://about.gitlab.com. [Zitat 1 10 2023].

5. Anhang

5.1. Export des Lerndesigns für das Projekt Bomberman

Siehe Datei LD_Bomberman.pdf

5.2. Export des Lerndesigns für das Projekt Tower Defense

Datei anzeigen LD_Tower_defense.pdf

5.3. Export des Lerndesigns für das Projekt Ameisen (Ants)

Datei anzeigen LD_Ants.pdf