



STUDY MATERIALS FOR FUTURE TEACHERS



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

Project	Object Oriented Programming for Fun
Project acronym	OOP4FUN
Agreement number	2021-1-SK01-KA220-SCH-00027903
Project coordinator	Žilinska univerzita v Žiline (Slovakia)
Project partners	Sveučilište u Zagrebu (Croatia) Srednja škola Ivanec (Croatia) Univerzita Pardubice (Czech Republic) Gymnázium, Pardubice, Dašická 1083 (Czech Republic) Obchodna akademia Povazska Bystrica (Slovakia) Hochschule für Technik und Wirtschaft Dresden (Germany) Gymnasium Dresden-Plauen (Germany) Univerzitet u Beogradu (Serbia) Gimnazija Ivanjica (Serbia)
Year of publication	2024

Disclaimer:

Funded by the European Union. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or Slovak Academic Association for International Cooperation. Neither the European Union nor the granting authority can be held responsible for them.

Table of contents

1. Information sheet.....	7
2. Introduction to Greenfoot environment.....	10
2.0. Exploring Game Development with Creativity	10
2.0.1. A teacher's Guide to Lesson Preparation	12
3. Class definition	15
Two teaching scenarios have been created within theClass definition thematic unit.	15
3.1. Exploring Classes and Objects through Game Development with Greenfoot	15
3.0.1. A teacher's Guide to Lesson Preparation	16
3.2. Creating Classes and Objects through Game Development with Greenfoot.....	20
3.2.1.A teacher's Guide to Lesson Preparation	21
4. Algorithm.....	25
Two teaching scenarios have been created within the Algorithm thematic unit.	25
4.0. Introduction to Algorithms in the Greenfoot Environment	25
4.1.1. A teacher's Guide to Lesson Preparation	26
4.1. Greenfoot Adventures: Unraveling Java Method Invocation, Documentation, and Application Control.....	29
4.1.1. A teacher's Guide to Lesson Preparation	30
5. Branching.....	35
Two teaching scenarios have been created within the Branching thematic unit.	35
5.0. Exploring Branching through Game Development with Greenfoot – Incomplete code branching.35	
5.0.1. A teacher's Guide to Lesson Preparation	36
5.1. Exploring Branching through Game Development with Greenfoot: Complete code branching	39
5.1.1. A teacher's Guide to Lesson Preparation	40
6. Variable and expressions.....	46
Five teaching scenarios have been created within the Variable and expressions thematic unit.	46
6.0. Introduction to Variables and Data Types in theGreenfoot Environment	46
6.0.1. A teacher's Guide to Lesson Preparation	47
6.1. Introduction to Operators and Expressions in the Greenfoot Environment.....	49
6.1.1. A teacher's Guide to Lesson Preparation	50
6.2. Introduction to Constructors in the Greenfoot Environment.....	54
6.2.1. A teacher's Guide to Lesson Preparation	55
6.3. Introduction to Attributes in the Greenfoot Environment	58
6.3.1. A teacher's Guide to Lesson Preparation	59
6.4. Introduction to Constructor Overloading in the Greenfoot Environment.....	62

6.4.1.	A teacher's Guide to Lesson Preparation	63
7.	Association	66
	Four teaching scenarios have been created within the Variable and expressions thematic unit.	66
7.0.	Greenfoot Objects on a Mission: Exploring Methods and Associations	66
7.0.1.	A Teacher's Guide to Lesson Preparation.....	68
7.1.	Greenfoot Objects on a Mission: Exploring Associations and Advanced Method Calls.....	74
7.1.1.	A Teacher's Guide to Lesson Preparation.....	76
7.2.	Greenfoot Objects on a Mission: Towers, Bullets, and Strategic Interactions.....	81
7.2.1.	A Teacher's Guide to Lesson Preparation.....	83
7.4:	Greenfoot Objects on a Mission: Bullets, Enemies, and Game Dynamics	92
7.2.2.	A Teacher's Guide to Lesson Preparation.....	94
8.	Inheritance	101
8.0.	Introduction to Inheritance in the Greenfoot Environment	101
8.0.1.	A teacher's Guide to Lesson Preparation	102
8.1.	Inheritance Concepts in theGreenfoot Environment (Part 1).....	106
8.1.1.	A teacher's Guide to Lesson Preparation	107
8.2.	Inheritance Concepts intheGreenfootEnvironment(Part 2).....	113
8.2.1.	A teacher's Guide to Lesson Preparation	114
8.3.	Inheritance Concepts in theGreenfoot Environment (Part 3).....	117
8.3.1.	A teacher's Guide to Lesson Preparation	118
9.	Encapsulation	122
9.1.	Exploring Encapsulation through Game Development with Greenfoot	122
9.1.1.	A teacher's Guide to Lesson Preparation	123
9.2.	Exploring Encapsulation through Game Development with Greenfoot	128
9.2.1.	A teacher's Guide to Lesson Preparation	129

List of tables

Figure 1. Task 1.1.....	14
Figure 2. Task 1.2.....	18
Figure 3. Task 1.3.....	19
Figure 4. Task 1.4.....	22
Figure 5. Task 2.3.....	32
Figure 6. Task 2.4.....	33
Figure 7. Task 2.5.....	33
Figure 8. Task 3.2.....	38
Figure 9. Task 3.3.1.....	41
Figure 10. Task 3.3.2.....	41
Figure 11. Task 3.4.....	42
Figure 12. Task 3.5: Configurations of custom setups of instances to predict movement of instance of class Enemy.....	43
Figure 13. Task 3.6: Configurations of tricky setups of instances to predict movement of instance of class Enemy.....	43
Figure 14. Task 3.7.....	44
Figure 15. Task - Turn in direction - 1.....	52
Figure 16. Task - Turn in direction - 2.....	53
Figure 17. Rename class MyWord to Arena.....	56
Figure 18. Create layout of Arena.....	57
Figure 19. Enemy.moveDelay.....	60
Figure 20- Movement of enemies respecting delay - 1.....	61
Figure 21 Movement of enemies respecting delay - 2.....	61
Figure 22. Parametric constructor of class Direction - 1.....	63
Figure 23. Parametric constructor of class Direction - 2.....	64
Figure 24. Task Overload constructors in class Direction - 1.....	65
Figure 25. Task Overload constructors in class Direction - 2.....	65
Figure 26. Task 5.3 - 1.....	70
Figure 27. Task 5.3 - 2.....	70
Figure 28. Task 5.3 - 3.....	70
Figure 29. Task 5.4.....	71
Figure 30. Task 5.5.....	72
Figure 31. Task 5.6.....	73
Figure 32. Task 5.7.....	77
Figure 33. Task 5.8.....	80
Figure 34. Task 5.9 - 1.....	84
Figure 35. Task 5.9 - 2.....	84
Figure 36. Task 5.11 - 1.....	86
Figure 37. Task 5.11 - 2.....	86
Figure 38. Task 5.14 - 1.....	89
Figure 39. Task 5.14 - 2.....	89
Figure 40. Task 5.14 - 3.....	90
Figure 41. Task 5.15 - 1.....	91
Figure 42. Task 5.15 - 2.....	91
Figure 43. Task 5.17 - 1.....	95
Figure 44. Task 5.17 - 2.....	96

Figure 45. Task 5.17 - 3.....	96
Figure 46. Task 5.17 - 4.....	96
Figure 47. Task 5.17 - 5.....	97
Figure 48. Task 5.17 - 6.....	97
Figure 49. Task 5.18 - 1.....	99
Figure 50. Task 5.18 - 2.....	99
Figure 51. Task 5.18 - 3.....	99
Figure 52. Task 5.18 - 4.....	100
Figure 53. Task 6.1 and 6.2 - 1.....	104
Figure 54. Task 6.1 and 6.2 - 2.....	104
Figure 55. Task 6.1 and 6.2 - 3.....	104
Figure 56. Task 6.3.....	105
Figure 57. Task 6.5 - 1.....	108
Figure 58. Task 6.5 - 2.....	108
Figure 59. Task 6.5 - 3.....	108
Figure 60. Task 6.7 - 1.....	110
Figure 61. Task 6.7 - 2.....	110
Figure 62. Task 6.7 - 3.....	111
Figure 63. Task 6.7 - 4.....	111
Figure 64. Task 6.7 - 5.....	112
Figure 65. Task 6.7 - 6.....	112
Figure 66. Task 6.8 - 1.....	115
Figure 67. Task 6.8 - 1.....	115
Figure 68. Task 6.9.....	116
Figure 69. Task 6.11 and 6.12 - 1.....	119
Figure 70. Task 6.11 and 6.12 - 2.....	119
Figure 71. Task 6.11 and 6.12 - 3.....	120
Figure 72. Task 6.11 and 6.12 - 4.....	121
Figure 73. Task 7.1.....	124
Figure 74. Task 7.2 and 7.3.....	125
Figure 75. Task 7.2 and 7.3.....	125
Figure 76. Task 7.4 - 1.....	127
Figure 77. task 7.4 - 2.....	127
Figure 78, Task 7.6.....	130
Figure 79. Task 7.7.....	131
Figure 80. Task 7.8 - 1.....	132
Figure 81. Task 7.8 - 2.....	132

1. Information sheet

The aim of this book is to present a course with materials that will help teachers in preparing materials to teach students to solve programming tasks using basics of object-oriented programming (OOP) following light OOP paradigm.

Students will learn to split given tasks among cooperating objects; to determine their competencies; and to implement designed model. The course does not require previous programming skills. It is taught in Java programming language. We use Greenfoot environment that utilizes Java programming language. Java is currently very popular and in praxis widely used programming language. Moreover, the Greenfoot presents the frame-based source code editor using Stride language. This opens possibilities for teachers who will want to use in this syllabus presented techniques with students of younger age. Greenfoot is very visual and from the beginning it makes it possible to create a visualized object, that is “alive” and can be interacted with. Therefore, the theoretical introduction is minimized, and students will start working from the very beginning.

The course explains light OOP concepts (such as encapsulation, inheritance, or association) on the creation of computer games, where these concepts are simply and intuitively utilized. The process of creating a computer game is based on teamwork and practically utilizes knowledge and skills from other areas of informatics and to it related subjects (work with multimedia and office software). The design of every computer game is open enough for students to expand the game individually and creatively. Moreover, the design leads to the proper utilization of acquired knowledge.

The book is focused to introduce innovative approach to teach programming, based on the solving of tasks using the object-oriented programming (OOP) paradigm. OOP is nowadays the dominant paradigm for application development. Therefore, it is proper for students to possess the knowledge and skills in this area. The subject presents development environment that utilizes different forms of source code editing (frame-based editing using simplified form as well as real source code writing) what makes it possible to teach students on different levels of prior technical knowledge and activity. With its simplicity and clarity this tool supports quick and intuitive comprehension of taught topics what has positive influence on students and their motivation.

Via programming of interactive games in graphical environment, the student will gain knowledge and skills, so that student will be able to:

- identify a problem,
- identify suitable objects to solve identified problem (object decomposition),
- design classes of objects, as well as their attributes and methods,
- identify and properly utilize objects relationships (association, inheritance),
- design an algorithm to solve problem and distribute it among cooperative objects,
- use source code elements (branching, loops) to implement designed algorithm,
- effectively use means for source code debugging,
- create simple application with graphical interface in the Greenfoot environment.

Learning outcomes of the subject are summarized as follows:

- understanding the basic principles of object-oriented programming,
- understanding the basics of algorithmization,
- understanding the syntax of the Java programming language,
- analyzing program execution based on the source code,

- the ability to create your own programs with the use of OOP.

A modern approach in designing lectures, especially those for elementary and highschool education is to define and to share *teaching scenarios*. Teaching scenarios (TS) “are perceived as a contemporary pedagogical approach which empowers individualisation of the teaching process by taking into consideration different student’s needs. TS based teaching is focused on relevant knowledge and skills for the students, including those of need for the digital society. Careful planning of TS can remedy possible pitfalls and shortcomings which might influence the teaching process.”

In the context of education and instructional design, TSs represent detailed descriptions or narratives that outline a specific instructional situation or context. These scenarios are often used in teacher training to simulate real-world teaching situations and thus we find it as the best tool to represent our innovative teaching and learning ideas. As teaching scenarios typically include information about the learning objectives, the content to be taught, the characteristics of the learners, the instructional methods employed, and the assessment strategies used, it can be aligned with the elements needed for our learning design artifacts as well.

In order to have a structured approach in defining several teaching scenarios we have defined the following template that would be filled with concrete data related to a specific learning scenario. Template contains a short description on how to define each element of the TS.

Title	Give learning scenarios a descriptive and an attention-grabbing title.
Learning objectives	Clearly state the intended learning outcomes. What should students know, understand, or be able to do by the end of the scenario?
Target audience	Specify the intended audience, grade level and pre-existing knowledge for which the learning scenario is designed.
Scenario duration	Estimate the time required to complete the learning scenario, including any specific timeframes for different activities. E.g. teacher introduction (5 min), research done by students individually (10 min), programming solution in team (20 min), presenting/discussion (10 min).
Materials&resources	List the materials, resources, and tools needed for both teachers and students. This could include textbooks, online and multimedia resources, software, etc.
Description	<ul style="list-style-type: none"> • Introduce the learning scenario, explain its purpose and relevance. • Outline the core activities that students will engage in to achieve the learning objectives. Include details such as discussions, hands-on activities, group work, competition, etc. • Specify how students will be organized, i.e. are they going to work

	<p>individually or in a team. How large are teams gonna be?</p> <ul style="list-style-type: none"> ● Explain what projects/problems/tasks will students be working on. Recommendation is to use problem-based or project-based approaches. Also, these should reflect real-life situations. ● Explain how projects/problems/tasks will be assigned to students (teams). ● If working in teams, provide details on how they are going to collaborate. ● Provide more details related to activities that students need to participate in. ● If e.g. flipped classrooms are used specify what part of the given topic students need to research by themselves.
Assessment	<p>Provide details related to how students' effort and knowledge will be assessed.</p> <ul style="list-style-type: none"> ● Who is going to assess the students: (1) teachers, (2) students their own work (self-assessment), (3) students each other's work (peer-assessment) ● What evaluation criteria will be used? ● How often will assessment be performed? ● etc.
Result dissemination	<p>Explain how students are going to disseminate their results to teachers and fellow students. E.g. students (all or subset) can present their results/solutions in front of the teacher and their peers, and then the comparison and discussion may follow.</p>

According to this our teaching scenario can be found on the following web address: [Course details \(learning-design.eu\)](https://learning-design.eu). Material that support this book can be found on the Moodle platform: <https://oop4fun.fon.bg.ac.rs/>

2. Introduction to Greenfoot environment

Greenfoot is a visual 2D educational software tool with a code editor to create games and simulations in Java programming language. Greenfoot is visual and interactive. The programmers are programmed in standard textual Java code, providing a combination of programming experience in a traditional text-based language with visual execution.

2.0. Exploring Game Development with Creativity

Title	Exploring Game Development with Creativity
Learning objectives	By the end of this session, students will not only have successfully installed Greenfoot and witnessed its capabilities through example projects but will also have engaged in collaborative, hands-on play within the development environment. This playful introduction sets the tone for an exciting exploration of game development, encouraging creativity, teamwork, and an enthusiastic approach to coding with Greenfoot.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including variables, functions, iteration and selection concepts.
Scenario duration	<ol style="list-style-type: none"> 1) Introduction (5 min) 2) Rush-hour challenge (10 min) 3) Playing games with teacher (30 min) 4) Team Formation and Project Assignment (5 min) 5) Team Collaboration and Coding (30 min) 6) Peer Review and Feedback (10 min) 7) Homework (30 min) 8) Competition grading (30 min)
Materials & resources	Greenfoot webpage and download instructions. Examples prepared by the teacher. Internet resources for identification of other examples.
Description	<p>In this 90-minute learning scenario, secondary school students will dive into the Greenfoot world by means of gamification, fun, research and teamwork.</p> <p>After the teacher introduces today's session, reflects on the previous session and sets challenging goals, the rush-hour challenge begins. Students are given the gamified assignment to find instructions, download and install Greenfoot (yet unknown development tool for them) on their computers. The first three students are given tokens of appreciation (badges, points, scores, sweets etc.).</p> <p>The second surprise for them is that in the next 30 minutes they will be playing games with the teacher. This is a teacher guided session on opening, compiling</p>

	<p>and running one-two simple example projects (on the introductory to medium level of complexity). This will show students the basic elements of the Greenfoot development environment and basic procedures for handling the project files and assets.</p> <p>Afterwards, the students will be grouped in teams (3-4 students each) and will be given a simple assignment. Teams should change “something” in the given example project to make the game surprising or fun. Team collaboration and coding (30 minutes) will have teams work collaboratively on trying to change something in the given examples. If they break the code beyond the line of being able to fix it on their own, they can ask for help from the teacher or can download the “start version” again. This will be a good example why we should use version control systems when coding.</p> <p>One or two teams will present their work for peer review and feedback and discuss the results with the teacher.</p> <p>At home for homework, each student should search for examples of Greenfoot games and should introduce his class to his favorite example by uploading a link, description of what makes it his favorite example and two-three screenshots of the development environment and running game. As part of the gamification and motivation via competition, each student should vote for the three best games (it is not allowed to vote for his own game). The winners are announced and awarded with tokens of appreciation (badges, points, scores, sweets etc.).</p>
Assessment	<p>This activity will enable teachers to give formative assessment feedback based upon the discussions and monitoring of students’ flipped classroom and teamwork.</p> <p>Gamification represents non formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p> <p>The peer-review assessment will be performed online as a part of a homework assignment. This will remind students of important aspects of the session, will make them install Greenfoot, give different examples and remind them of what was done during the class, which will increase the overall achievement of learning outcomes.</p>
Result dissemination	<p>To disseminate their results to teachers and fellow students the usual setup of the Learning management system (e.g. Moodle will be used). The students can continue the discussion on the topic on the forum provided to them via the Learning management tool.</p>

2.0.1. A teacher's Guide to Lesson Preparation

1. Introduction

Objective: Introduce students with programming languages (visual and textual), integrated development environments and source code.

Concepts to Discuss: programming languages, integrated development environments and source code

Activity: The teacher makes a short introduction for today's lesson. The teacher introduces concepts such as:

- programming language
- integrated development environments (programming tools) and
- source code.

The teacher presents examples of source code in different programming languages such as Java, C, and Python, but also demonstrates programs in Scratch or another visual language.

The teacher does not go "in depth" but tries to explain these concepts using a simple example, for example, trying to compare learning a programming language with learning a native or foreign language.

- Natural languages have their own grammar and spelling; the same is the case with programming languages. In writing, we follow some grammar and spelling rules, similarly when we write a program in a programming language.
- When we write a story in our native language, we use a notebook, a pen as a tool that helps us do it, similarly when we write a program in a programming language, our "tool" for that is integrated development environment
- The result of writing can be a story written on paper, while the result of coding is a program that we create, which we refer to as source code

The teacher is free to make an introduction with his own example.

2. Rush-hour challenge

Objective: Investigate Greenfoot and installation instructions.

Concepts to Discuss: Greenfoot, installation instructions

Activity: The teacher assigns the task to the students to find out what Greenfoot is. The teacher requires from the students to find the installation instructions for Greenfoot. Students work on the task, after that the teacher presents the students how to find, download, and run the installation instructions for Greenfoot. [Instructionson how to download and install Greenfoot the teacher can find on Moodle platform.](#)

3. Playing games with teacher

Objective: Start different Greenfoot projects.

Concepts to Discuss: Greenfoot project (web based and standalone projects)

Activity: The teacher gives the students a task to look at and find examples of projects made in Greenfoot on the Internet. Students are looking for projects while teachers follow their work.

The teacher presents the students how they can find examples of finished projects made in the Greenfoot environment. For example, in the Google search engine, projects can be found based on the keywords 'Greenfoot Java project example' or similar.

The teacher explains students that there are two types of Greenfoot projects:

- one that can be run in a web browser, and
- another that can be downloaded and then opened and run from the Greenfoot environment.

The teacher presents projects:

- Projects that can be run directly in a web browser
- Projects that can be run in Greenfoot after downloading.

The teacher can download some example projects or access some projects by accessing the links.

4. Team Formation and Project Assignment

Objective: Include the student in project base learning with simple assignment task.

Concepts to Discuss: -

Activity: The teacher creates teams, prepares a task and sets a project for the students to work on. **Some of the examples of the task can be found on the Moodle.** The teacher chooses one representative project (not very complex) and for that project defines a task (problem) that the students should solve.

5. Team Collaboration and Coding

Objective: Introduce the student to create Greenfoot project. Students are working on the assignment task.

Concepts to Discuss: create Greenfoot project

Activity: The teacher starts Greenfoot and presents the students how to create a project. The teacher assigns the task to the students to create a Greenfoot project.

Task 1.1: Create a new project and give it a proper name (e.g. **TowerDefense**) and save it to a proper location.

The teacher emphasizes to the students that project (identical to the project that was just made) can be:

download from git¹ by typing the git checkout command

Commit: [9046f5353d857dcc112abd92d7b7170abcc64a80](#))

- download from git, but ZIP file from the following address

<https://oop4fun.fon.bg.ac.rs/>

The teacher emphasizes to the students that in today's class, they will not work on the project they have just created, but will do so from the next class, and that they will now work on existing projects.

¹Students must have Git installed on their computer.

The students take on the task and solve the problem.

Solution:

When the solution is created three new classes are made – World, descendant class MyWorld and Actor. World and Actor classes cannot be changed. The class MyWorld has a public constructor which makes a new world size 600x400px by default, with a cell size of 1x1 pixels.

```
3 public class MyWorld extends World
4 {
5
6     /**
7     * Constructor for objects of class MyWorld.
8     *
9     */
10    public MyWorld()
11    {
12        // Create a new world with 600x400 cells
13        //with a cell size of 1x1 pixels.
14        super(600, 400, 1);
15    }
16 }
```

Figure 1. Task 1.1

6. Peer Review and Feedback

Objective: Discussion on the students proposed solution.

Concepts to Discuss: -

Activity: The teacher monitors the students' work, and if necessary, gives them guidelines (instructions). After the end, the teacher chooses one team to show their solution. The teacher discusses the proposed solution with the students.

7. Homework

Objective: Give tasks to the students to explore the Greenfoot projects.

Concepts to Discuss: -

Activity: The teacher defines the task that the students should solve on one of the existing projects.

8. Competition grading

Objective: Include the students in evaluation process of the students' projects.

Concepts to Discuss: -

Activity: Students present their project. The teacher requires everyone to participate in the evaluation of the presented projects by sending links to the students to fill Google form. The students score on the three best teams.

After presentations, the teacher gives a summary of the student projects. The teacher shares his impressions regarding the students' work, indicating whether he is satisfied with it, whether they met his expectations, or if they exceeded them.

3. Class definition

Two teaching scenarios have been created within theClass definition thematic unit.

3.1. Exploring Classes and Objects through Game Development with Greenfoot

Title	Exploring Classes and Objects through Game Development with Greenfoot
Learning objectives	By the end of this session, students will be able to understand basic concept of the object and class . Understanding of the examined concepts will be discussed in the context of the game development, encouraging creativity, teamwork, and an enthusiastic approach to coding with theGreenfoot tool.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including object and class .
Scenario duration	<ol style="list-style-type: none"> 1) Object (10 min) 2) Identification of objects and their properties (15 min) 3) Class, instance, inheritance (15 min) 4) Orientation in Greenfoot: World, Actor, MyWorld (10 min) 5) Class constructor(10 min) 6) Task 1.2(15 min) 7) Image settings (10 min) 8) Task 1.3(15 min)
Materials & resources	Greenfoot webpage and download instructions. Examples prepared by the teacher. Internet resources for identification of other examples.
Description	<p>In this 90-minute learning scenario, secondary school students are introduced to programming principles related to object and class through the lens of game development using the Greenfoot tool.</p> <p>The session starts with a 10-minuteintroduction by the teacher, which introduces students to the world of object-oriented programming by explaining the concept of an object and their properties in real life.</p> <p>After that, the teacher gives a rush-hour challenge (10 minutes) to the students who must identify the objects and their properties based on the task's textual description. After that, the teacher together with the students comes up with a solution for the task (5 minutes).</p> <p>In thecontinuation of the lesson, the teacher explains the difference between the class and the object. At the highest level of abstraction, it explains the concept of inheritance (15 minutes).</p> <p>The tutorial starts the Greenfoot environment and explains the Word, Actor, and</p>

	<p>MyWord classes (10 minutes).The teacher explains and presents the source code that was generated and stands behind the classes World, Actor, MyWorld after creating the projects (10 minutes).</p> <p>The teacher starts task 1.2 and shows the students how to create a World for the application they need to build (10minutes).The teacher explains how to set up a picture for a particular class and works together with the students on task 1.3 (20 minutes).</p>
Assessment	Gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.
Result dissemination	To disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. Moodle will be used). The students can continue the discussion on the topic on the forum provided to them via the Learning management tool.

3.0.1. A teacher's Guide to Lesson Preparation

1. Object

Objective: Introduce the students with the object concept by real-life examples.

Concepts to Discuss: object and object characteristics.

Activity: The teacher introduces the term object. The teacher should make this concept more relatable to students by using real-life examples. For instance, the teacher could ask students about their name, height, birthdate, eye color...The teacher asks these questions to prompt students to consider how they differ from each another, preparing them for a later question: How students differ from each other?

The teacher concludes that each of us (teacher, students, pupils) has characteristics by which we differ from each other and emphasizes that each of us is actually "one" object. The teacher explains the concept of a trait as a characteristic that each of us possesses, along with a specific value for each trait unique to each individual. Hence, objects vary from each other based on the values they possess for their respective characteristics.

2. Identification of objects and their properties

Objective: Involve students to identify objects and their properties.

Concepts to Discuss: objects and their properties

Activity: The teacher gives the task to the students to identify objects and their properties based on the text.

3. Class, instance, inheritance

Objective: Acquire knowledge about class and instance. Make the difference between class and instance.

Concepts to Discuss: class, instance

Activity: In order to explain the concept of class, class instances (objects) and inheritance the teacher gives real life examples.

The teacher asks the students questions to make them to see the difference between the class and the object. Teacher guided discussion on recognized objects and their classification in classes.

4. Creating an instance of the world in Greenfoot

Objective: Introduce the world in Greenfoot.

Concepts to Discuss: instance of the world in Greenfoot

Activity: The teacher starts the Greenfoot environment and creates a simple project. The teacher uses this project to explain how students create introduced concept into Greenfoot.

The teacher noticed that every project created in the Greenfoot environment contains 3 classes, **Word**, **Actor**, and **MyWorld**.

The teacher presents the **MyWorld** class and its role.

The teacher emphasizes that the background of each application created in Greenfoot consists of cells that represent a single matrix. The teacher shows how to define the background size (matrix dimension) and the size of each matrix cell.

The teacher explains that the objects that appear on the table ("scene") are on one of these cells. The teacher shows the source code behind each of these classes.

5. Prepare world

Objective: Involve students to work on project task.

Concepts to Discuss: world in Greenfoot

Activity: The teacher gives the task to the students. **Task 1.2:** Create a world of size 10x10 cells. Each cell should be 75 pixels in size.

Teacher follows the students work and at the end asks one students to show his solution and describe it.

Description of the solution:

Edit source code of class MyWorld (double-click on it) to create a world of size 24x12 cells. Each cell should be 50 pixels in size.

Commit: [a593cd4a92d0fa0db78275614c3e41a2e96b4e57](https://github.com/username/repo/commit/a593cd4a92d0fa0db78275614c3e41a2e96b4e57)

Solution: Adjust the MyWorld class so that size of the world is 24x12 with a cell size of 50x50 pixels.

```

3 public class MyWorld extends World
4 {
5     public MyWorld()
6     {
7         // Create a new world with 24x12 cells
8         // with a cell size of 50x50 pixels.
9         super(24, 12, 50);
10    }
11 }

```

Figure 2. Task 1.2

6. Class constructor

Objective: Introduce the class constructor

Concepts to Discuss: constructor

Activity: The teacher presents source code and introduce the concept constructor.

7. Image settings

Objective: Set the image of the world in the Greenfoot project.

Concepts to Discuss: Image settings for the World class.

Activity: The teacher explains to the students that the background of Greenfoot application (or world) can be an image. The teacher explains to the students that the background can either be a single image covering the entire world area or an image sized to match the cell's dimensions.

The teacher shows how to set a image on the MyWorld class. Students follow the teacher's instructions and work together step by step.

8. Prepare world graphics

Objective: Introduce students with background of the World class

Concepts to Discuss: background of the World class

Activity: The teacher gives assignments to students, **Task 1.3:** Create proper image for the World background and set it. The teacher explains to the students what he expects them to do. The teacher can download final project that have already been prepared by professor, and show them. The teacher provides the students a link or git command to download the initial project, which they need to update with this task (feature). The initial project can be download from git repository.

The students do the task independently or in the group. The teacher monitors the students' work.

The teacher demonstrates the solution step by step, while the students follow the instructions. Description of the solution:

- Find or create a proper image for the world background. You may either use prepared images (select item Set image... from the context menu of class MyWorld) or custom image (copy image into subfolder images of your project folder and select it using the same way as described before).
- As a background you may use sole image that will cover whole world's area (compute needed size of the image with regards to the world's size) or smaller one, that will be repeatedly copied (use square image with the size of the cell).

Commit: [1184980643db082cfdd6bde9984bceaddf010d49](#)

Solution: The appearance of the World with the background image applied is shown in the next picture.

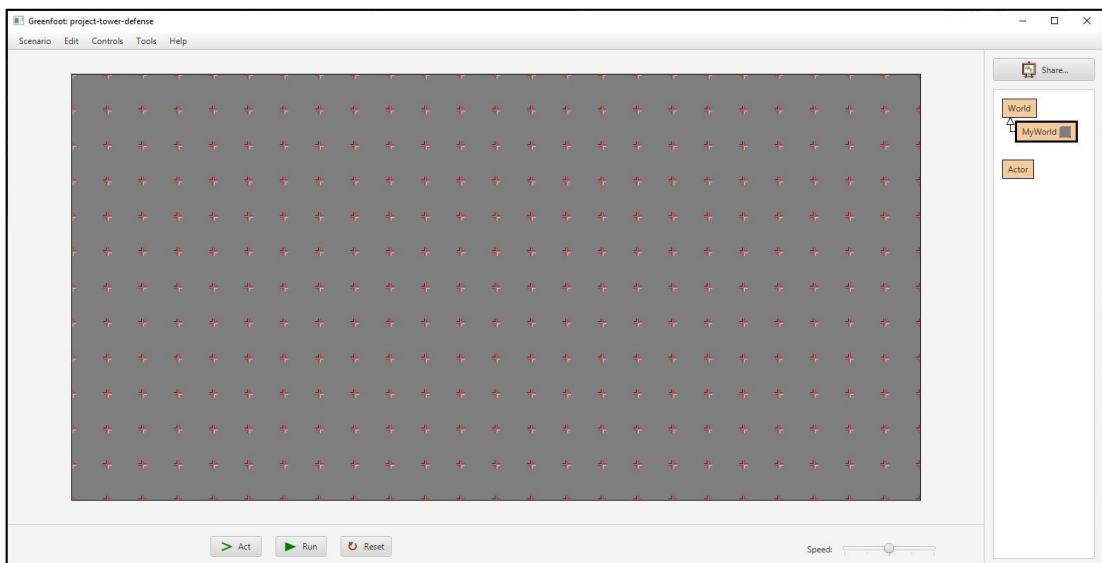


Figure 3. Task 1.3

3.2. Creating Classes and Objects through Game Development with Greenfoot

Title	Creating Classes and Objects through Game Development with Greenfoot
Learning objectives	By the end of this session, students will understand basic concepts of the object, class, class properties and methods.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including object, class, class properties and methods .
Scenario duration	<ol style="list-style-type: none"> 1) Basic concepts(25 min) 2) Task 1.4 (10 min) 3) Task 1.5(30 min) 4) Interface of object (5 min) 5) Message and method (15 min) 6) Task 1.6 (30 min) 7) Theory revision (15 min)
Materials & resources	Greenfoot webpage and download instructions. Examples prepared by the teacher. Internet resources for identification of other examples.
Description	In this session, students will progress through several structured tasks to deepen their understanding of object-oriented programming concepts. Initially, they will spend 25 minutes creating a class named Enemy, followed by a focused 15-minute task to define the attributes and methods within this class. Next, they will allocate 30 minutes to instantiate an object of the Enemy class, applying their knowledge of object creation and initialization. The concept of an object's interface will then be explored in 5 minutes, emphasizing the definition of what operations an object can perform. Subsequently, students will delve into the concepts of messages and methods over 15 minutes, learning how objects communicate through method invocation. The session will dedicate another 30 minutes to practical application, where students will send messages to their instantiated Enemy instance, reinforcing their understanding of object behavior. Finally, a 15-minute theoretical revision will recapitulate key concepts such as objects, classes, instances, internal state, identity, messages, and methods, ensuring a comprehensive grasp of the session's learning objectives.

Assessment	Gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.
Result dissemination	To disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. Moodle will be used). The students can continue the discussion on the topic on the forum provided to them via the Learning management tool.

3.2.1.A teacher's Guide to Lesson Preparation

1.1. Basic concept

Objective: Determine knowledge about class, its properties, and objects.

Concepts to Discuss: class, properties, and objects

Activity: During the introduction to the lesson, the teacher reviews previously covered concepts with the students. Through discussion, the teacher clarifies the concepts of class, properties of classes, and objects.

The teacher assigns a written task to the students, instructing them to identify specific classes, their properties, associated objects, and respective values within a provided text.

The teacher selects a student from the class to present their solution. During this activity, other students participate by offering their opinions.

Professor step by describe how to create class in Greenfoot environment. Students follow the teacher's instructions and work together step by step.

1. Task 1.4

Objective: Involve student to work on the project.

Concepts to Discuss: create class

Activity: The teacher gives assignments to students, Task 1.4: Create class Enemy and put appropriate image for it.

The teacher explains to the students what he expects them to do. The teacher can download the final project that has already been prepared by the teacher and show them. The teacher provides the students a link or git command to download the initial project, which they need to update with this task (feature). The initial project for this task can be downloaded from git repository.

The students do the task independently or in the group. The teacher monitors the students' work. The teacher demonstrates the solution step by step, while the students follow the instructions. Description of the solution:

Create an enemy. Enemy will march towards player's orb to damage and eventually destroy it. Create a new subclass of class Actor (select item Actor by right mouse click and choose

“Newsubclass”... from the context menu of class Actor). Give it proper name (Enemy) and image. The teacher should explain the convention that class should start with capital letters and the whole Java name convention.

Commit: [4981400623729c3d112b54454b6e6151e18426bf](https://github.com/4981400623729c3d112b54454b6e6151e18426bf)

Solution: The added Enemy class extends the Actor class and has defined *act()* class.

```
1 import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
2
3 /**
4  * Write a description of class Enemy here.
5  *
6  * @author (your name)
7  * @version (a version number or a date)
8  */
9 public class Enemy extends Actor
10 {
11     /**
12      * Act - do whatever the Enemy wants to do. This method is called whenever
13      * the 'Act' or 'Run' button gets pressed in the environment.
14      */
15     public void act()
16     {
17         // Add your action code here.
18     }
19 }
```

Figure 4. Task 1.4

Task1.5

Objective: Introduce the state of the object. Lear students to create instance of the class in the Greenfoot.

Concepts to Discuss: state of the object, instance

Activity: The teacher explains the concept of the state of the object with a simple example. For example, the teacher is positioned in the classroom at a specific distance from the front door. This distance determines their location, with steps forward or backward altering their proximity to the door. Hence, the teacher's position relative to the front door can be quantified by a variable that changes over time. Thus, the teacher's position is defined by the distance from the front door, illustrating how the state of an object—in this case, the teacher—is characterized by the value of its attribute (distance) at any given moment.

Teacher describe task 1.5 how to create instance of the class Enemy. Teacher opens the last version of the project while students follow the teacher's instructions and work together step by step.

Teacher creates an instance of class Enemy (select item by mouse right click new Enemy () from the context menu of class Enemy, put instance into world by left mouse click on desired position). Investigate its internal state (select item place into the World by right mouse click and choose Inspect from the context menu of the created instance).

The teacher asks students to create a new instance and put it in another position to compare internal states of two created instances.

2. Interface of object

Objective: Introduce the students with interface as a set of action that we can perform on some object.

Concepts to Discuss: interface

Activity: The teacher introduces the concept of an interface to students through simple examples. For instance, if we observe a person and the activities they perform during the day (such as waking up, having breakfast, going to work) without delving into specific details—like how they wake up (whether by an alarm clock, phone, or a parent), what and where they eat breakfast, or their mode of transportation to work—the collection of these activities can be likened to an interface. An interface defines what actions objects of a certain class can perform but does not specify how those actions are carried out or executed. We don't define what he eats for breakfast, how he went to work (on foot, by car or by bus) or how he woke up (whether someone called him or the clock wake him up).

3. Message and method

Objective: Introduce students with method concept.

Concepts to Discuss: class method

Activity: The teacher introduces the concept of the method. To explain the concept of a method, the teacher connects the concepts of properties (characteristics, attributes) and changing values for those properties.

Example: If we consider the class Person and the property age, its value increases by one every year, consistently on the same day. On the other hand, if we look at properties like height and weight, these characteristics change frequently—people grow and their weight fluctuates.

Example: If we observe the movement of a person from point A to point B, and describe this movement in steps—such as stepping forward, turning left by 45 degrees, taking 8 steps forward, turning right by 30 degrees, and taking 5 more steps forward—these individual actions can be grouped into what we call a method.

The teacher shows the students in Greenfoot how they can look at the methods a class has that can be called on a specific object.

The teacher shows the methods defined in the Actor class. The teachershowshow to callmethods on anobject.

4. Task 1.6

Objective: Introduce students how to call method on instance object.

Concepts to Discuss: call methods

Activity: Teacher describes task 1.6 send a message to instance of class Enemy (calling the method). Teacher opens the last version of the project while students follow the teacher's instructions and work together step by step.

Teachersends messages to the instance of class Enemy so it will move into position [12, 6] and it will be facing down (by right click to the object in the Worls, select inherited from

Actor and then select method setLocation(int, int)). Describe the students what will happen with the instance. How was the internal state of respective instance affected?

The teacher demonstrates to the students how methods are defined. He creates a setPosition(int x, int y) method for setting the Actor to specific coordinates. The teacher emphasizes that this method is equivalent to setLocation(int x, int y) and stresses the importance of checking if a method already exists before defining a new one to avoid duplication. It's emphasized that method names should clearly indicate their purpose and functionality, allowing immediate understanding from the name alone. The teacher also underscores that method names should be concise and provides examples of well-defined, poorly-defined, and improperly-defined methods. Methods that the user can perform (call) are visible by right clicking on an object.

Professor describes how to define method. Students follow the teacher's instructions and work together step by step.

The teacher asks the students to define the method by which the Actor is lowered (decreases the y-coordinate) and the method by which it is raised (increases the y-coordinate). Teacher follow the students how they work on the task and give instruction one by one if it is necessary.

5. Theory revision

Objective: Summarize the concept that covered in the lecture.

Concepts to Discuss: object, class, instance, internal state, identity, message, method.

Activity: Teacher summarizes the concept that covered in the lecture.

4. Algorithm

Two teaching scenarios have been created within the Algorithm thematic unit.

4.0. Introduction to Algorithms in the Greenfoot Environment

Title	Introduction to Algorithms and Algorithmic Thinking
Learning objectives	By the end of the scenario, students should have a solid understanding of algorithms and algorithmic thinking, be proficient in designing and implementing basic algorithms, and be able to apply algorithmic concepts to solve a variety of problems effectively.
Target audience	Secondary school students attending the OOP4Fun course. Students should have a basic programming knowledge, including variables, functions, iteration and selection concepts, proficiency in logical reasoning and problem-solving skills. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none">1) Introduction to basic algorithms as a sequence of steps (15 min)2) Task 2.1 (20 min)3) Algorithm and its properties (15 min)4) Task 2.2 (25min)5) Algorithmisation (15 min)
Materials & resources	The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.
Description	<p>In this learning scenario, secondary school students will delve into the realm of algorithms and algorithmic thinking. The session begins with a 15-minute introduction aimed at familiarizing students with basic algorithmic concepts, highlighting their significance in problem-solving.</p> <p>Following the introduction, students will engage in a 20-minute task where they are tasked with writing a simple algorithm to address a specific problem. This hands-on activity allows students to apply the concepts introduced earlier and hone their algorithmic skills.</p> <p>Subsequently, a 15-minute segment will be dedicated to discussing algorithms and their properties. Topics covered will include correctness, efficiency, and scalability, emphasizing the importance of clear and precise instructions in algorithm design.</p> <p>Building upon their understanding, students will spend the next 25 minutes crafting a more general algorithm for a slightly complex problem. This task challenges students to think abstractly and critically, applying algorithmic principles to tackle real-world scenarios.</p>

	<p>In the final 15-minute segment, students will engage in algorithmization, analyzing and refining their algorithms. This process involves identifying potential improvements, optimizing efficiency, and ensuring the robustness of the algorithms.</p> <p>Throughout the session, students will work individually or in small groups, fostering collaboration and peer learning. By actively participating in writing and analyzing algorithms, students will develop critical thinking skills and computational problem-solving abilities.</p> <p>At the conclusion of the session, students will have a deeper understanding of algorithms and algorithmic thinking, equipping them with essential skills for approaching complex problems systematically and effectively.</p>
Assessment	The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.
Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.

4.1.1. A teacher's Guide to Lesson Preparation

1. Introduction to basic algorithms as a sequence of steps

Objective: Introduce the algorithm.

Concepts to Discuss: basic of the algorithm

Activity:The teacher introduces the concept of algorithm through examples from real-life. For example, the teacher asks the students what they do in the morning from the time they wake up until the time they come to school. The teacher then asks the students if they know how to make a pizza or a hot sandwich, or if any of them know a recipe for making a dish or a cake.

The teacher connects the process of making a meal or cake with making a program and emphasizes that just as there is a recipe for preparing a meal, there is also a "recipe" for making a program called an algorithm.

The teacher concludes that an algorithm is a set of steps that define how a program is executed.

The teacher further explains that this set of steps does not necessarily always have to be executed sequentially, one after the other, that there are some steps in the algorithm that can be executed depending on some condition. The teacher asks the students to give an example of such a case (for example, if we are writing an algorithm for preparing a hot sandwich, if we do not have an ingredient, for example ham, but we have a similar ingredient, we can replace it, or go to the store and get missing ingredient).

The teacher explains to the students that some steps in the algorithm can be repeated several times. The teacher asks the students to give an example of an algorithm in which the steps are repeated several times.

2. Task 2.1

Objective: Introduce students how to write simple algorithm.

Concepts to Discuss: algorithm

Activity: The teacher assigns the students task 2.1 to write on paper the procedure by which they describe how a pedestrian crosses the street.

At the beginning, the teacher does not give additional instructions to the students but follows how they think and work. If someone asks a question that is important for the description of the instructions for pedestrians crossing the street, he praises him and emphasizes why this information is important.

After some time, the teacher asks the students if they have paid attention to where the pedestrian crosses the street, whether it is a marked place to cross or not. Also, the teacher asks the students whether they have considered whether there is a traffic light at the pedestrian crossing or not.

At the end, the teacher chooses a few students who should read their instructions for crossing the street.

3. Algorithm and its properties

Objective: Introduce students with algorithm properties

Concepts to Discuss: algorithm properties

Activity: The teacher further explains to the students about the properties of an algorithm. The teacher explains to the students that algorithms can also be shown graphically and gives an example of algorithms that are graphically represented.

4. Task 2.2.

Objective: Write a more general algorithm

Concepts to Discuss: write algorithm

Activity: Make a general algorithm for the preparation of a hot drink. Think about what the inputs of such an algorithm need to be for it to be general.

5. Algorithmisation

Objective: Give more examples to the students and include students to define its own

Concepts to Discuss: Algorithm

Activity: The teacher explains to the students that even in mathematics there are certain algorithms that we use in solving problems. The teacher asks the students if they can give an example.

The example that the teacher explains to the students is an example of calculating the value of an arithmetic expression that has several mathematical operations, where it is necessary to respect the rules of priority of performing mathematical operations.

Also, the teacher gives some other examples, such as assembling new furniture that has been purchased and coming with instructions that describe how to assemble this furniture. An example can be the instructions we receive through the GS device when we want to go from point A to point B via navigation.

The teacher asks the students to write on the paper their own algorithm, after which some of them present the result.

4.1. Greenfoot Adventures: Unraveling Java Method Invocation, Documentation, and Application Control

Title	Greenfoot Adventures: Unraveling Java Method Invocation, Documentation, and Application Control
Learning objectives	By the end of the scenario, students should have a solid understanding of Java method invocation, particularly focusing on the 'act()' and 'move()' methods within the Greenfoot environment. They should be proficient in effectively utilizing the 'this' keyword to reference current objects within a class context. Additionally, students should demonstrate the ability to call methods within a class, comprehending the syntax and parameters required for method invocation. They should also showcase proficiency in applying method calling techniques to effectively solve interactive game development tasks. Furthermore, students should grasp the importance of code documentation and be able to document Java code effectively, ensuring clarity and readability. Lastly, they should master application control techniques within Greenfoot, enabling precise manipulation and interaction with game elements to create engaging and functional game mechanics.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including iteration and selection concepts. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none"> 1) Explanation of act () method (10 min) 2) Explanation of move() method (20 min) 3) Introducing keyword this (5 min) 4) Task 2.3 (10 min) 5) Explanation of Autocompleting (5 min) 6) The importance of code documentation (15 min) 7) Task 2.4 (5 min) 8) Task 2.5 (5 min) 9) Task 2.6 (10 min) 10) Task 2.7 (20 min) 11) Discussion: Algorithm, properties, algorithmisation, Greenfoot buttons (5 min)
Materials & resources	The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.
Description	In this 105-minute learning scenario, secondary school students will embark on a journey to master Java method invocation, code documentation, and application control within the Greenfoot environment. The session kicks off with a 15-minute exploration of the 'act()' method, followed

	<p>by a 10-minute dive into the 'move()' method, essential components in Greenfoot programming.</p> <p>Students will then spend 5 minutes delving into the significance of the 'this' keyword in referencing current objects within a class context.</p> <p>Following this, a 10-minute task awaits students, challenging them to practice calling methods within a class, applying the syntax and parameters required for method invocation.</p> <p>A 5-minute explanation of autocompletion features in Greenfoot environment follows, focusing on enhancing coding efficiency.</p> <p>For the next 15 minutes, students will grasp the importance of code documentation, understanding how clear and concise documentation enhances code readability and maintainability. In a 5-minute task, students will add documentation to their code, ensuring clarity and comprehensibility for themselves and others. Building upon this task, students will spend an additional 5 minutes adding more detailed documentation to their code. In a 10-minute task, students will explore and read documentation added by their peers, gaining insights into different coding styles and approaches.</p> <p>Finally, students will spend 20 minutes exploring application control techniques within Greenfoot, enabling precise manipulation and interaction with game elements to create engaging and functional game mechanics.</p> <p>Throughout the session, students will work individually or in small groups, fostering collaboration and peer learning. By actively participating in writing and analyzing code, students will develop critical thinking skills and computational problem-solving abilities.</p> <p>At the session's end, students will emerge with a deeper understanding of Java method invocation, code documentation, and application control in Greenfoot, equipping them with essential skills for game development and beyond.</p>
Assessment	Gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.
Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. Moodle will be used). The students can continue the discussion on the topic on the forum provided to them via the Learning management tool.

4.1.1. A teacher's Guide to Lesson Preparation

1. Explanation of act() method

Objective: Explain the using of the act() method. Explain calling the method from the other method.

Concepts to Discuss:method act()

Activity:Teacher open the last version of the **TowerDefinse** project.The teacher puts instances of the **Enemy** class to position 0,0. The teacher asks the students what will happen when we call the **act()** method on an instance of the **Enemy** class. Is it the expected behavior?

Teacher ask student to help him to done task, to move the object of the **Enemy** class two cells in the current direction when we call **act()** method.

The teacher shows the students how to do this task (adding the **move(int)** method call inside the **act()** method), while the students follow the teacher's instructions and give him ideas.

2. Explanation of **move()** method

Objective: Explain the **move()** method and how to move forward and backward.

Concepts to Discuss: methods for movements

Activity:The teacher explains the **move(int)** method. The teacher puts instances of the **Enemy** class in different position and calls method passing the positive values, for example 1 or 3.Teacher ask the students, what they think, what will happen if they call the **move()** method and enter negative values, for example -1.

The teacher assigns the students a task to write a **backward()** method that moves an object 1 step backward. Is it possible to move object 2 steps back and X steps? What we need to do?

3. Introducing keyword **this**

Objective: Introducing keyword **this**

Concepts to Discuss: keyword **this**

Activity:Teacher introducing keyword **this**.

4. Task 2.3.

Objective: Learn students how to write method to move object vertically changing the value of y axis.

Concepts to Discuss: method invocation

Activity:Teacher asks the students how to move the object vertically, up or down?The teacher asks student to find suitable method for up/down move by right click on an object and selecting Inherited from Actor. Student should recognize following methods: **turn**, **setRotation**, **setLocation**, **getLocation**, **getRotation**.The teacher assigns the students the task to write the method **up()** and **down()**. The teacher monitors the students' work and at the end explains together with them how to implement these methods. The methods **up()** and **down()** change the value of coordinate y, increase and decrease respectively. Here teacher can introduce the method parameters but without details.

Commit: [7ba327ebeba6a13be68d9d21cc7e74b0da376132](https://github.com/7ba327ebeba6a13be68d9d21cc7e74b0da376132)

Solution: The **act()**method should be changed in order to move the Enemy.

```

3 public class Enemy extends Actor
4 {
5     /**
6     * Act - do whatever the Enemy wants to do. This method is called whenever
7     * the 'Act' or 'Run' button gets pressed in the environment.
8     */
9     public void act()
10    {
11        this.move(2);
12    }
13 }

```

Figure 5. Task 2.3

5. Explanation of Autocompleting

Objective: Introduce students with autocomplete in Greenfoot environment

Concepts to Discuss: autocomplete (CTRL+SPACE)

Activity: Teacher describe students how to find method on the object that they can call if they forgot the name or if they just start coding and want to „ask“ Greenfoot environment to help them to finish writing code faster.

6. The importance of code documentation

Objective: Introduce students with comments and documentation

Concepts to Discuss: comments and documentation, explore the documentation window.

Activity: Teacher introduces students with a statement in the code that is not part of the execution (comments, documentation comments).

Teacher need to emphasize differences between comments and documentations (as a special type of comments).

Teacher presents students source code of the **Enemy** class and after that generated html document that describes documentation of the **Enemy** class. Teacher describes here that there is some rule that we need to follow when write documentation for our class or methods.

Teacher assigns task to student to explore how to write documentation for Java class and methods.

7. Task 2.4.

Objective: Introduce student with method documentation tag

Concepts to Discuss: method documentation tag,

Activity: Teacher assigns task students to add a documentation comment for the **act()** method.

Commit: [68b1c82c7df2c7826f2d3f78373498569adab7e9](https://github.com/68b1c82c7df2c7826f2d3f78373498569adab7e9)

Solution: Add the comment above the *act()* method which describes what the method does.


```

2
3 public class Enemy extends Actor
4 {
5     /**
6      * Enemy moves two cells in its current direction
7      */
8     public void act()
9     {
10        this.move(2);
11    }
12 }

```

Figure 6. Task 2.4

8. Task 2.5.

Objective: Introduce student with class documentation tag

Concepts to Discuss: class documentation tag

Activity: Teacher assigns task students to add a documentation comment for the **act()** method.

Edit the documentation comment of the **Enemy** class. Add the version of the class and its author and see the changes in the html generated page.

Commit: [1a7a9f83c5271a7c0dfa46ce3b1ee65682b0c5e5](https://github.com/1a7a9f83c5271a7c0dfa46ce3b1ee65682b0c5e5)

Solution: The version and the author in the comment above the Enemy class should be modified.

```

3 /**
4  * Class represents enemy.
5  *
6  * @author Michal Varga
7  * @version 1.0
8  */
9 public class Enemy extends Actor
10 {
11     /**
12      * Enemy moves two cells in its current direction.
13      */
14     public void act()
15     {
16        this.move(2);
17    }
18 }

```

Figure 7. Task 2.5

9. Task 2.6.

Objective: Learn students to explore documentation of the class in order to meet methods

Concepts to Discuss: explore documentation

Activity:Teacher assigns task students to explore the documentation window and read documentation for the Actor and Window class. Teacher emphasize important of reading documentation in order to discover methods that can be useful.

10. Task 2.7.

Objective: Introduce students with Greenfoot buttons

Concepts to Discuss: Greenfoot buttons

Activity:The teacher continue to work on the last version of the project. The teacher asks the students to add two instance object of the **Enemy** class and call the **act()** method on each instance.

Teacher describes Greenfoot**Act** button. Teacher clicks the Greenfoot**Act** button located on the main window. The teacher asks a question for them to conclude and explain what happened.

Also, the teacher asks the student to click on the **Run** button and to conclude what happened.

The teacher asks the students to click the **Reset** button and explain what happened. After that, the teacher explains what needs to be done so that every time the **Reset** button is clicked, two objects of the **Enemy** class appear on the board at positions (0,3) and (3,3).

The teacher should explain to the students that if he wants the objects to appear on the stage every time the Reset button is clicked, it is necessary to change the constructor of the World class, so that those objects are created in the constructor and placed in the desired positions.

11. Discussion: Algorithm, properties, algorithmisation, Greenfoot buttons

Objective: The teacher summarizes the lesson.

Concepts to Discuss: method form movements, Greenfoot buttons, documentation

Activity:The teacher summarizes the lesson. The professor can emphasize here important of the code documentation, rules for method and class naming convention.

5. Branching

Two teaching scenarios have been created within the Branching thematic unit.

5.0. Exploring Branching through Game Development with Greenfoot – Incomplete code branching

Title	Exploring Incomplete Branching through Game Development with Greenfoot
Learning objectives	Topic covers incomplete branching (multiple branching is intentionally omitted). Basics of Actor's World perception is introduced. Students will be capable of writing code using conditions. Upon finishing this topic, students will learn how to use simple if-else statements and learn how to make decisions in their code controlling how their game behaves. They will gain a basic knowledge of Java programming language, learning the necessary syntax and to analyze and understand code, helping them understand why their game behaves a certain way and how to fix issues. Students will be able to create their own game projects using what they've learned about object-oriented programming.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including variables, functions, iteration and selection concepts. Students should be introduced to Greenfoot in general.
Scenario duration	<ol style="list-style-type: none">1) Introduction (5 minutes)2) Code explanation (15 minutes)3) Incomplete branching(10 minutes)4) Observing the players' state (10 min)5) Adding world edge detection (10 min)
Materials & resources	The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from GitHub/Gitlab. Internet resources.
Description	<p>The session begins with a 15-minute code explanation of the turn() method, which sets the stage for understanding incomplete code branching. This discussion is led by the teacher and is aimed at bringing students to a common understanding of the method's role in the actors world.</p> <p>Following this, a 10-minute segment is dedicated to acquiring basic concepts related to incomplete branching. This acquisition phase is crucial for students to grasp the foundational principles before delving into code writing.</p> <p>Next, the students engage in a 10-minute investigation task, where they observe the player state within their code, fostering a deeper understanding of the program's flow based on the source code.</p> <p>The subsequent 10-minute production phase involves adding world edge</p>

	detection to their project, adding behavior and setting up the game.
Assessment	<p>This activity will enable teachers to give formative assessment feedback based upon the discussions and monitoring of students' flipped classroom.</p> <p>The peer-review assessment will be performed online as a part of a homework assignment. This will remind students of important aspects of the exercise, will make them critically assess other students' work, will give them insights into good or not so good solutions of their peers etc., and will increase the overall achievement of learning outcomes.</p> <p>The work in the team-project that the students are working on will also use these learning outcomes and knowledge.</p>
Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of GitHub/Gitlab and Learning management system (e.g. Moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.

5.0.1. A teacher's Guide to Lesson Preparation

1. Introduction

Objective: The teacher discusses with the students the concepts that were covered in the previous lesson. Teacher introduces goals for this teaching session.

Concepts to Discuss: algorithm, code documentation

Activity: In the introductory part of the lesson, the teacher reviews with the students the concepts adopted from the previous lesson. The teacher then introduces the new material that will be studied in today's class. To illustrate today's lesson, the teacher presents an example of an algorithm with simple branching. One example could be the algorithm for crossing the street at a pedestrian crossing where there is no traffic light. The pedestrian does not cross the street immediately but first ensures there are no vehicles coming from the left or right. If there are no vehicles, then the pedestrian crosses the street.

2. Code explanation

Objective:

Concepts to Discuss:

Activity:The teacher downloads the latest version of the project:

- From Moodle platform
- From git repository

The teacher creates and places an Enemy class object somewhere on the board. It explains some methods of the Actor class:

- move(int)
- turn(int)

- setRotation()

While explaining the methods, the teacher also shows how certain properties of the class are changed (for example, the position of the object on the board, ie the x and y values). The teacher discusses with the students how to supplement the act() method so that every time the act() method is called, the Enemy class object should move two steps forward.

3. Incomplete branching

Objective:

Concepts to Discuss: branching, incomplete branching

Activity: The teacher continues to work on the project. The teacher places an Enemy class object on the board. The teacher explains to the students how they can check if the object is in the upper half of the board and displays the message "Found".

The teacher explains to the students the showText() method, which is used to display text.

4. Observing the players' state

Objective:

Concepts to Discuss: internal state of the instance

Activity: The teacher creates an instance of the **Enemy** class and places it in the center of the board. The teacher opens a window with the internal state of the instance and positions it so that it is visible while the application is running. Then run the application and observe how the values of the **x**, **y** and **rotation** attributes in the **Enemy** class change when call different methods. How do these values change as you move (up, down, left, and right) and turn?

5. Adding world edge detection (10 min)

Objective:

Concepts to Discuss: world edge detection

Activity: The teacher discusses with the students how they can determine whether an object is on the edge or not. For example, as far as we know the dimensions of the table, based on the position (x,y) it can be determined whether the object is on the edge of the table or not.

Teacher puts the instance somewhere in the world (but not on the edge), invoke method *IsAtEdge()*. He discuss with students what happened. The teacher move the enemy to the edge and examine the result of method *IsAtEdge()*, which should now return true.

The teacher explains the method *isAtEdge()*.

Task 3.2: Teacher assigns task to students to add code to the body of the act() method to rotate the enemy 180° by calling the setRotation() property, when it reaches the edge of the world.

The teacher discusses with the students how an object that has reached the edge can continue its movement:

- backwards (without turning)
- backwards (with turning)

Together with the students, the teacher completes the program code of the `act()` method. Now, run `act()` method and discuss what happens with the enemy once it reaches the edge of the World.

Commit: [4927c3ff7eb39b51ba2738f2ab500fd6c32e3bb4](https://github.com/4927c3ff7eb39b51ba2738f2ab500fd6c32e3bb4)

Solution: In the `act()` method of the `Enemy` class, detection of the World's edge should be added.

```
9 public class Enemy extends Actor
10 {
11     /**
12      * Enemy moves two cells in its current direction.
13      */
14     public void act()
15     {
16         this.move(2);
17
18         if (this.isAtEdge())
19         {
20             this.turn(180);
21         }
22     }
23 }
```

Figure 8. Task 3.2

5.1. Exploring Branching through Game Development with Greenfoot: Complete code branching

Title	Exploring Complete Branching through Game Development with Greenfoot
Learning objectives	Topic covers incomplete and complete branching (multiple branching is intentionally omitted). Basics of Actor's World perception is introduced. Students will be capable of writing code using conditions. Upon finishing this topic, students will learn how to use simple if-else statements and learn how to make decisions in their code controlling how their game behaves. They will gain a basic knowledge of Java programming language, learning the necessary syntax and to analyze and understand code, helping them understand why their game behaves a certain way and how to fix issues. Students will be able to create their own game projects using what they've learned about object-oriented programming.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including variables, functions, iteration and selection concepts. Students should be introduced to Greenfoot in general.
Scenario duration	<ol style="list-style-type: none"> 1) Add classes Direction and Orb (30 min) 2) Collision detection explanation (30 min) 3) Task 3.4 4) Task 3.5 5) Task 3.6 6) Code explanation: Complete branching (15 minutes) 7) Task 3.7 (20 minutes) 8) Task 3.8 (30 minutes) 9) Revision (5 minutes)
Materials & resources	The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from GitHub/Gitlab. Internet resources.
Description	<p>The session starts with a 30-minute practice session to add classes Direction and Orb. This helps students comprehend the object-oriented nature of Java and the importance of structuring code properly. A 20-minute code explanation follows, focused on the concept of collision detection, which helps students understand object interactions within their game environment. Then, students spend 10 minutes adding collision detection to their project in another production task, utilizing their understanding of branching.</p> <p>Investigative tasks of 15 minutes each are assigned next, where students are tasked to predict enemy movement in both a custom and a tricky setup, enhancing their problem-solving and analytical skills. Another 15-minute discussion session is focused complete branching, ensuring that students can</p>

	<p>differentiate between incomplete and complete code structures.</p> <p>A 20-minute production task engages students in using full branching with collision detection, aiming to consolidate their learning by applying complex concepts in a practical setting. The session concludes with a challenging 30-minute investigative task where students again predict enemy movement, this time with the added experience from the previous tasks, thus applying their cumulative knowledge.</p> <p>The session ends with 5-minute theory revision of the learned concepts.</p>
Assessment	<p>This activity will enable teachers to give formative assessment feedback based upon the discussions and monitoring of students' flipped classroom and teamwork.</p> <p>The peer-review assessment will be performed online as a part of a homework assignment. This will remind students of important aspects of the exercise, will make them critically assess other students' work, will give them insights into good or not so good solutions of their peers etc., and will increase the overall achievement of learning outcomes.</p> <p>The work in the team-project that the students are working on will also use these learning outcomes and knowledge.</p>
Result dissemination	<p>In order to disseminate their results to teachers and fellow students the usual setup of GitHub/Gitlab and Learning management system (e.g. Moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.</p>

5.1.1. A teacher's Guide to Lesson Preparation

1. Add classes **Direction** and **Orb**

Objective: Understand how to add new class in the project.

Concepts to Discuss: class

Activity: Teacher adds assignment to students to work on Task 3.3. Teacher follows the students' activities, and in the end, he asks one student to present his work. The student describes and presents his work.

Task 3.3: Create two new classes, descendants of the Actor class. The first class will be Direction class and the second class will be Orb. Prepare suitable (max. 50x50 pixel) images in a graphical editor. Then assign these images to the newly created classes.

Commit: [4ed6b37e6d481181d8b340639aa03391406b6c2e](#)

Solution: Two new classes should be added – Orb and Direction. The following code for the class Orb is generated, when the class is created in the Greenfoot.


```

9 public class Orb extends Actor
10 {
11     /**
12     * Act - do whatever the Orb wants to do. This method is called whenever
13     * the 'Act' or 'Run' button gets pressed in the environment.
14     */
15     public void act()
16     {
17         // Add your action code here.
18     }
19 }

```

Figure 9. Task 3.3.1

Similar code is generated when the Direction class is created.

```

9 public class Direction extends Actor
10 {
11     /**
12     * Act - do whatever the Direction wants to do. This method is called whenever
13     * the 'Act' or 'Run' button gets pressed in the environment.
14     */
15     public void act()
16     {
17         // Add your action code here.
18     }
19 }

```

Figure 10. Task 3.3.2

The appropriate images should be set for both classes.

2. Discussion: Collision detection explanation

Objective: Explain the collision detection.

Concepts to Discuss: collision detection

Activity: The teacher put an instance of the **Enemy** class on the World, and an instance of the **Direction** class in the same row. The teacher adds code to the **act()** method so that the object moves one step forward.

The teacher explains to the students how to determine whether two or more objects ("characters") on the World are in the same position (on the same cell). The teacher explains the method: **isTouching()**.

The teacher and students modify the **act()** method of the **Enemy** class to ensure that the enemy rotates 90° clockwise when it is in the same cell that contains an instance of the **Direction** class.

Together with the students, the teacher observes what happens with the **rotation** attribute.

3. Task 3.4

Objective: Students understood how to determinate if two objects are in the same cell.

Concepts to Discuss: -

Activity: Teacher adds assignment to students to work on task 3.4. Teacher follows the students activities, and in the end he asks one student to present his work. The student describes and presents his work.

Task 3.4: Add code to the **act()** method of the **Enemy** class to ensure that:

- the player turns 90° counter clockwise when he enters a cell the contains an instance of the **Orb** class.

Commit: [968e6f195e3def25e11bc41b664ba1715f7da11d](https://github.com/968e6f195e3def25e11bc41b664ba1715f7da11d)

Solution: The *act()* method should be modified in order to turn the player when the collision occurs.

```
9 public class Enemy extends Actor
10 {
11     /**
12      * Enemy moves two cells in its current direction.
13      */
14     public void act()
15     {
16         this.move(2);
17
18         if (this.isAtEdge())
19         {
20             this.turn(180);
21         }
22
23         if (this.isTouching(Direction.class))
24         {
25             this.turn(90);
26         }
27
28         if (this.isTouching(Orb.class))
29         {
30             this.turn(-90);
31         }
32     }
33 }
```

Figure 11. Task 3.4

4. Task 3.5

Objective: Understand the objects movements.

Concepts to Discuss: objects movements

Activity: Teacher adds assignment to students to work on task 3.5. Teacher follows the students activities, and in the end he asks one student to present his work. The student describes and presents his work.

Tasks 3.5: Prepare different configurations, inspiration can be found in the figures below. Guess how the enemy will move? Run the application. Does your prediction match what you observe? What caused differences in prediction and reality?

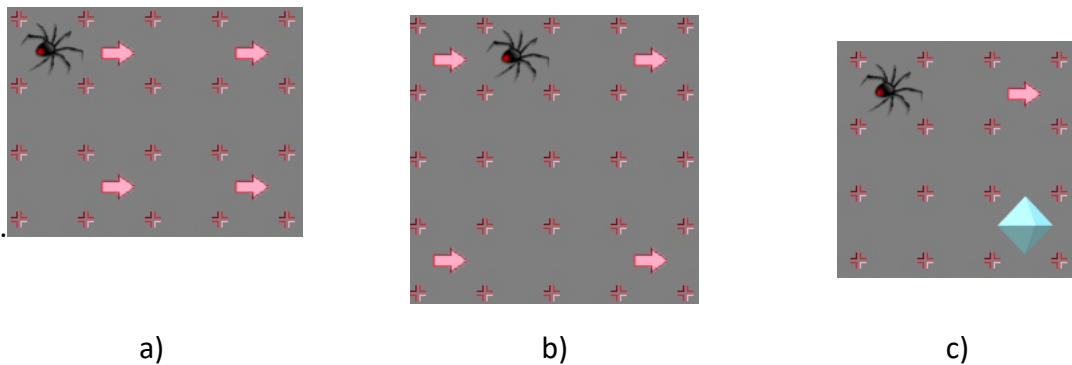


Figure 12. Task 3.5: Configurations of custom setups of instances to predict movement of instance of class Enemy

5. Task 3.6

Objective: Student understood the objects movements.

Concepts to Discuss: objects movements

Activity: Teacher adds assignment to students to work on task 3.6. Teacher follows the students activities, and in the end he asks one student to present his work. The student describes and presents his work.

Tasks 3.6: Prepare the situation as depicted in the figure below. Guess how the enemy will move? Run the application. Does your prediction match what you observe? What caused differences in prediction and reality?

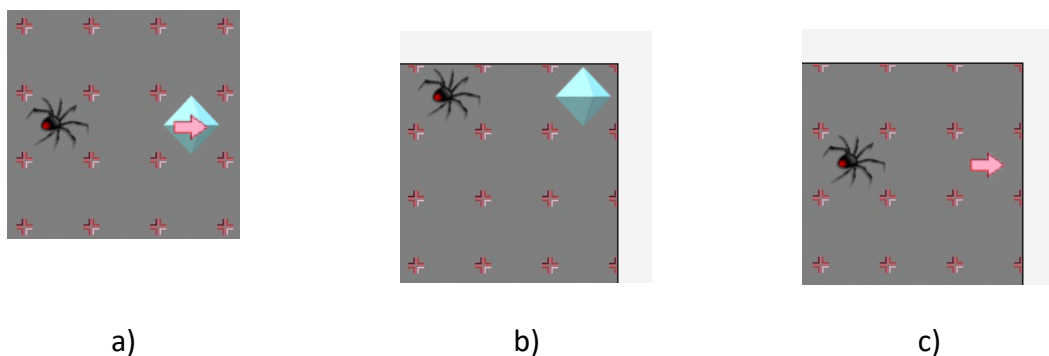


Figure 13. Task 3.6: Configurations of tricky setups of instances to predict movement of instance of class Enemy

6. Code explanation: Complete branching

Objective: Teacher taught the students when and how to use if-then-else and switch statement.

Concepts to Discuss: if-then-else statement, nested if statement, switch statement

Activity: The teacher assigns the students the task of describing on paper how a pedestrian crosses the street. The teacher monitors how the students solve the task. At some point the teacher emphasizes to the students to pay attention to whether or not there is a traffic light at the place where the street is crossed. If in the meantime, that is before the teachers emphasize this, one of the students asks this question or something similar, he publicly

praises him and emphasizes that it is important that before coding we should always analyze the problem first and identify all cases that can occur. Here students should understand complete and nested branching.

Teacher will ask the students to put objects Orb and Distance class to the left, right to or right place – to the different borders of the World. Students will describe Enemy “bad” behavior. Teacher should explain that there are two conditions satisfied at one situation. The enemy is touching Orb/Description class and is touching the edge. By drawing the situation on the table and paper students will recognize that complete and nesting branching is needed and then students change the code of enemy behavior.

7. Task 3.7

Objective: Student understood complete branching nested if and switch statement .

Concepts to Discuss: nested if statement, switch statement

Activity: Teacher adds assignment to students to work on task 3.7. Teacher follows the students activities, and in the end he asks one student to present his work. The student describes and presents his work.

Task 3.7. After the code of the method **act()** of class **Enemy** so that it will use full branching. You will create a nested conditions. Make the detection of the edge the most important check, then check the touch of the instance of **Direction** class and lastly check the touch of the instance of **Orb** class.

Commit: [f017de8b49d4fc77f62afac4d842429560bcfb8b](https://github.com/f017de8b49d4fc77f62afac4d842429560bcfb8b)

Solution: The act() method should be modified to handle additional conditions.

```
9 public class Enemy extends Actor
10 {
11     /**
12     public class Enemy extends Actor
13     {
14         /**
15         * Enemy moves two cells in its current direction.
16         */
17         public void act()
18         {
19             this.move(2);
20
21             if (this.isAtEdge())
22             {
23                 this.turn(180);
24             }
25             else
26             {
27                 if (this.isTouching(Direction.class))
28                 {
29                     this.turn(90);
30                 }
31                 else
32                 {
33                     if (this.isTouching(Orb.class))
34                     {
35                         this.turn(-90);
36                     } // this.isTouching(Orb.class)
37                 } // this.isTouching(Direction.class)
38             } // this.isAtEdge()
39         }
40     }
41 }
```

Figure 14. Task 3.7

8. Task 3.8

Objective: Predict enemy movement on custom setup

Concepts to Discuss: object behaviour

Activity: The teacher puts objects arbitrarily in the World, and the students explain their movement and behavior (independently or in pairs).

Tasks 3.8: Run through tasks **Error! Reference source not found.** and **Error! Reference source not found.** again. What changed?

9. Revision

Objective: The teacher summarizes the lesson.

Concepts to Discuss: concept of the branching

Activity: The teacher summarizes the lesson.

6. Variable and expressions

Five teaching scenarios have been created within the Variable and expressions thematic unit.

6.0. Introduction to Variables and Data Types in the Greenfoot Environment

Title	Introduction to Variables and Data Types in the Greenfoot Environment
Learning objectives	By the end of this session, students will be able to understand data types and variables. Understanding of the examined concepts will be discussed in the context of the game development, encouraging creativity, teamwork, and an enthusiastic approach to coding with the Greenfoot tool.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including variables and data types. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none">1) Introduction (10 minutes)2) Variable identification (5 minutes)3) Data types (15 minutes)4) Declaration of variables (10 minutes)5) Initialization of variables (5 minutes)
Materials & resources	The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.
Description	<p>In this 45-minute learning scenario, secondary school students are introduced to programming principles related to data types and variables through the lens of game development using the Greenfoot tool. The session starts with a 10-minute introduction by the teacher, establishing the context related to the previous sessions, which creates the basis for variable introduction and definition.</p> <p>This is followed by a 5-minute scenario, during which students and teachers research and identify variables for their game. Taking into account that each variable must be of a certain type, in the next 15 minutes various data types are presented.</p>

	<p>The core activities encompass a 10-minute teacher-guided session, during which students declare variables for their game, emphasizing the importance of variable names and types. This is followed by a 5-minute session related to variable initialization, in which variable values are defined. In this context, behavior of objects in the game can be changed (e.g., rotation of an object, movement of an object).</p> <p>The students will continue working on the game that was explained and started in the previous sessions. As a result, at the end of the session novel concepts related to variables and data types are introduced.</p>
Assessment	The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.
Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.

6.0.1. A teacher's Guide to Lesson Preparation

1. Introduction

- In the introduction section context related to the previous sessions is established. The teacher introduces the term *variable*.

2. Variable identification

Objective: Variable identification through discussion, emphasizing the role of variables in programming.

Concepts to Discuss: Variables and values.

Activity:

- Teacher introduces the term *variable*,
- Students can be asked to research and identify variables for their game,
- The variables can be discussed by the teacher and peers,
- In this scenario, variable type can be omitted (or discussed in general).

3. Data types

Objective: Understanding the concept of data types, recognize their real-world applications, and focusing on the variable types needed for game development.

Concepts to Discuss: Variables, Data types, Real-world data type examples, Variable types for the game.

Activity:

- Teacher introduces the term *data type*,

- Examples from real-world can be discussed (e.g., integer numbers can be related to number of currently present students, decimal numbers can be related to a product price, text type can be related to instant messaging text, etc.),
- Data types are considered in the context of the Greenfoot Environment and Java programming language,
- Detailed discussion related to variable types required for the game.

4. Declaration of variables

Objective:Applying the concepts of data types and variables, and declare variables necessary for game development.

Concepts to Discuss: Variables, Data types, Variable types for game.

Activity:

- Data types are considered in the context of the Greenfoot Environment and Java programming language,
- Teacher should explain the difference between declaration and initialization of variables:
 - In variable declaration, variable of a specific data type is declared, but value may/may not be present,
 - Analogy can be introduced (e.g., labeled box for a specific cookie type, but without cookie inside the labeled box),
- Declaration of Game-required variables.
- Additional examples can be considered. For example, if *act()* method is considered, variable for displaying text can be declared.

5. Initialization of variables

Objective: Applying the concepts of data types and variables, and initialize variables required for the game.

Concepts to Discuss: Variables, Data types, Variable values for game.

Activity:

- Based on previously presented data types, their data values and data ranges are introduced,
- Data values and data ranges are considered in the context of the Greenfoot Environment and Java programming language,
- Teacher should explain the difference between declaration and initialization of variables:
 - In variable initialization, variable of a specific data type is declared and can be initialized at the same time (this is called: *initial value*), and can be changed in the following code,
 - Analogy can be introduced (e.g., cookie of a specific cookie type is placed in the previously defined labeled box),
- Initialization of game-required variables.
- Additional examples can be considered. For example, if *act()* method is considered, variable for displaying text can be initialized.

6.1. Introduction to Operators and Expressions in the Greenfoot Environment

Title	Introduction to Operators and Expressions in the Greenfoot Environment
Learning objectives	By the end of this session, students will be able to understand the concept of operators. The session introduces various operator types (i.e., arithmetic operators, boolean operators, relational operators), as well as their respective expressions. In addition, object expression and reference variable will be introduced. The examined concepts will be discussed in the context of the game development, encouraging creativity, teamwork, and an enthusiastic approach to coding in the Greenfoot environment.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including iteration and selection concepts. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none"> 1) Operators (15 minutes) 2) Arithmetic operators and expressions (10 minutes) 3) Boolean operators (15 minutes) 4) Relational operators (10 minutes) 5) Boolean expressions (10 minutes) 6) Object expression (5 minutes) 7) Reference variable and its null value (15 minutes) 8) Task - Turn in direction (15 minutes)
Materials & resources	<p>The textbook from the OOP4Fun project.</p> <p>Resources from OOP4Fun project.</p> <p>Project source code from Github/Gitlab.</p> <p>Internet resources.</p>
Description	<p>During this 95-minute learning session, secondary school students are introduced to programming concepts related to operators and expressions within the context of game development using the Greenfoot tool. The session starts with a 15-minute introduction by the teacher, providing background information from previous sessions, which creates the basis for operator introduction and definition.</p> <p>This is followed by a 10-minute scenario, during which arithmetic operators and expressions are discussed. Arithmetic operators are used for arithmetic operations. In this context, various operators and expressions in the Greenfoot environment are practically explained and discussed.</p> <p>The next 15-minute session introduces boolean operators. These are logical operators used to manipulate boolean values. This is followed by a 10-minute session related to relational operators which are used for comparing values. Based on the previous sessions, the next 10-minute scenario discusses boolean expressions in the context of the Greenfoot environment.</p>

	<p>The next 5-minute session focuses on object expression, while the following 15-minute section discusses reference variables.</p> <p>Finally, this is followed by a 15-minute teacher-guided session, during which students solve task related to turn in direction for their game. The feedback provided by the teacher and peers is included in this session.</p> <p>The students will continue working on the game project initiated in the previous sessions. Consequently, by the end of the session, they will be introduced to concepts related to operators and expressions.</p>
Assessment	<p>The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p> <p>The state of the project opens possibilities for home assignments. In this context, more classes, expressions and values can be introduced to achieve additional behavior (e.g. teleports, tunnels, etc.). These concepts can be discussed with students and the respective implementation can be assigned as homework.</p>
Result dissemination	<p>In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.</p>

6.1.1. A teacher's Guide to Lesson Preparation

1. Operators

Objective: Understanding the concept of operators, relate them to real-life scenarios, and learn about various types of operators.

Concepts to Discuss: Variables, Data types, Operators, Real-world examples of operators.

Activity: The teacher introduces the term *operator*. The teacher should make this concept more relatable to students by using real-life examples (e.g., buying products at the market). Afterwards, the teacher introduces various operator types.

2. Arithmetic operators and expressions

Objective: Understanding the concept of arithmetic operators, relate them to real-life scenarios, and learn about various arithmetic operators.

Concepts to Discuss: Variables, Data types, Operators, Real-world examples of arithmetic operators.

Activity: Teacher can explain operators already known from other courses (e.g., math and math arithmetic operators). These operators are considered in the context of the Greenfoot Environment and Java programming language. Teacher discusses various terms: operator, operand, operator precedence. Additional examples can be considered. Additional example may include defining local variables to retrieve and manipulate an entity's x-position and y-position, thereby changing its position by increasing the variable's values.

3. Boolean operators

Objective: Understanding the concept of boolean operators, relate them to real-life scenarios, and learn about various boolean operators.

Concepts to Discuss: Variables, Data types, Operators, Real-world examples of boolean operators.

Activity: Teacher can explain operators already known from other courses (e.g., math and math Boolean operators). These operators are considered in the context of the Greenfoot Environment and Java programming language. The teacher discusses various terms: operator, operand, operator precedence. Additional examples may include defining local variables to check if an entity's x-position is equal to its y-position, using a boolean operator to determine if the entity is on a diagonal.

4. Relational operators

Objective: Understanding the concept of relational operators, relate them to real-life scenarios, and learn about various relational operators.

Concepts to Discuss: Variables, Data types, Operators, Real-world examples of operators.

Activity: Teacher can explain operators already known from other courses (e.g., math and math relational operators), These operators are considered in the context of the Greenfoot Environment and Java programming language. Teacher discusses various terms: operator, operand, operator precedence. Additional examples may include defining local variables to check if one entity's y-position is below another's, using relational operators to determine positional relationships between entities.

5. Boolean expressions

Objective: Understanding the concept of boolean expressions, relate them to real-life scenarios, and learn about various boolean expressions.

Concepts to Discuss: Variables, Data types, Operators, Real-world examples of boolean expressions.

Activity: The teacher can explain Boolean expressions in the context of previously presented operators. These expressions are considered in the context of the Greenfoot Environment and Java programming language. Teacher discusses operator, operand, and operator precedence in the context of boolean expressions. Additional examples may be considered. For example, boolean expressions can be used to verify that entity's position is inside defined arena's dimension of the game.

6. Object expression

Objective: Understanding the concept of object expressions, relate them to real-life scenarios, and learn about various object expressions.

Concepts to Discuss: Variables, Data types, Operators, Real-world examples of object expressions.

Activity: Teacher can explain object expression in the context of object-oriented design. These expressions are considered in the context of the Greenfoot Environment and Java programming language. The teacher discusses operator, operand, operator precedence,

and class casting in the context of object expressions. Additional examples can be explored. For instance, comparing references of two object entities to check if they overlap.

7. Reference variable

Objective: Understanding the concept of reference variables, relate them to real-life scenarios, and apply them in game development.

Concepts to Discuss: Variables, Data types, Operators, Real-world examples of reference variables.

Activity: Teacher can explain reference variables in the context of object-oriented design. These reference variables are considered in the context of the Greenfoot Environment and Java programming language. The teachers should explain null reference value.

8. Task - Turn in direction

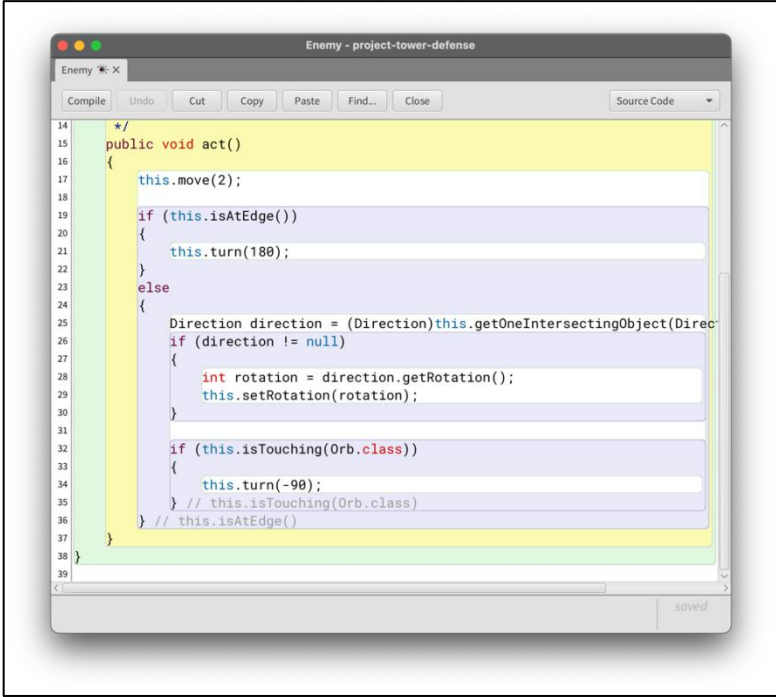
Objective: Understanding the concept of variables, data types, operators, expressions, and utilize them in the game development.

Concepts to Discuss: Variables, Data types, Operators, Expressions.

Activity: Teacher should discuss Enemy's **act()** method. Teacher explains how to use local variables in the code, for example to use "rotation" variable. Teacher should describe difference between **this.rotation** and **rotation**. Teacher should describe **getOneIntersectingObject(_cls_)** method behavior. It is used and instance is stored in proper local variable (class casting is required). If there is no intersection object it returns null value. Based on the performed boolean evaluation, an appropriate acting is performed (i.e., rotating or turning). The results are discussed by the teacher and peers.

Commit: [97dddc4beba40ac785c7413bb245ba849cd956d2](https://github.com/97dddc4beba40ac785c7413bb245ba849cd956d2).

Solution: Implementation of **act()** method in the Enemy class.



```
14  */
15  public void act()
16  {
17      this.move(2);
18
19      if (this.isAtEdge())
20      {
21          this.turn(180);
22      }
23      else
24      {
25          Direction direction = (Direction)this.getOneIntersectingObject(Direc
26          if (direction != null)
27          {
28              int rotation = direction.getRotation();
29              this.setRotation(rotation);
30          }
31
32          if (this.isTouching(Orb.class))
33          {
34              this.turn(-90);
35          } // this.isTouching(Orb.class)
36          // this.isAtEdge()
37      }
38  }
39  }
```

Figure 15. Task - Turn in direction - 1

```
12  * Enemy moves two cells in its current direction.
13  * If it steps on the Direction instance, it adjusts rotation accordingly.
14  */
15  public void act()
16  {
17      this.move(2);
18
19      if (this.isAtEdge())
20      {
21          this.turn(180);
22      }
23      else
24      {
25          Direction direction = (Direction)this.getOneIntersectingObject(Dire
26          if (direction != null)
27          {
28              int rotation = direction.getRotation();
29              this.setRotation(rotation);
30          }
31
32          if (this.isTouching(Orb.class))
33          {
34              this.turn(-90);
35          } // this.isTouching(Orb.class)
36      } // this.isAtEdge()
37  }
```

saved

Figure 16. Task - Turn in direction - 2

6.2. Introduction to Constructors in the Greenfoot Environment

Title	Introduction to Constructors in the Greenfoot Environment
Learning objectives	By the end of this session, students will be able to understand the concept of constructors. The session introduces basic theoretical concepts related to constructors, as well as various code explanations and tasks. The examined concepts will be discussed in the context of the game development, encouraging creativity, teamwork, and an enthusiastic approach to coding in the Greenfoot environment.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge and basic object-oriented knowledge. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none"> 1) Basic concepts of constructors (10 minutes) 2) Code explanation (20 minutes) 3) Task: Rename class MyWord to Arena (5 minutes) 4) Task: Create layout of Arena (30 minutes)
Materials & resources	<p>The textbook from the OOP4Fun project.</p> <p>Resources from OOP4Fun project.</p> <p>Project source code from Github/Gitlab.</p> <p>Internet resources.</p>
Description	<p>During this 65-minute learning session, secondary school students are introduced to concepts related to constructors within the context of game development using the Greenfoot tool. The session starts with a 10-minute introduction by the teacher, providing basic theoretical concepts of constructors.</p> <p>This is followed by a 20-minute scenario, during which various code examples in the Greenfoot environment are practically explained and discussed.</p> <p>The next 5-minute session is related to task. Previously defined class MyWorld should be renamed. In this context, new name should be defined, i.e. class name should be Arena. It is important to note that constructor of the class should also be renamed.</p> <p>Finally, this is followed by a 30-minute teacher-guided session, during which students solve task related to layout of Arena. The feedback provided by the teacher and peers is included in this session.</p> <p>The students will continue working on the game project initiated in the previous sessions. Consequently, by the end of the session, they will be introduced to concepts related to constructors.</p>
Assessment	The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.

	The state of the project opens possibilities for home assignments. In this context, more classes, expressions and values can be introduced to achieve additional behavior (e.g. teleports, tunnels, etc.). These concepts can be discussed with students and the respective implementation can be assigned as homework.
Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.

6.2.1. A teacher's Guide to Lesson Preparation

1. Basic concepts of constructors

Objective: Understanding the concept of constructors.

Concepts to Discuss: Constructors, Classes, Objects, Keywords: *super, new, this*.

Activity: The teacher introduces the term *constructor* within the context of Class and Object concepts in Object-Oriented Programming (OOP). Constructors are used to initialize a concrete instance (i.e., an object) of a class.

2. Code explanation

Objective: Understanding the concept of constructors, relate them to real-life scenarios.

Concepts to Discuss: Constructors, Classes, Objects, Methods, Parameters, Keywords: *super, new, this*, Real-world examples of constructors.

Activity: The teacher should discuss constructors within the context of Class and Object OOP concepts: constructors are used to initialize concrete instances of a class. In addition, constructors are always invoked and can be defined either implicitly or explicitly. There are default constructors (which are implicitly defined) as well as parameterized and non-parameterized constructors (which are explicitly defined by a programmer). The differences between parameterized and non-parameterized constructors should also be discussed. To make this concept more relatable to students, the teacher should use real-life examples.

3. Task: Rename class MyWord to Arena

Objective: Renaming the MyWorld class.

Concepts to Discuss: Constructors, Classes, Objects.

Activity: The previously defined class MyWorld should be renamed. In this context, a new name should be chosen, specifically Arena. Additionally, the constructor of the class should also be renamed from **MyWorld()** to **Arena()**.

Commit: [aaf73c9bfd9f76a2a1e504f5e78d2976f1cada12](#)

Solution: MyWorld class was renamed to Arena class. Constructor of the class is also renamed from MyWorld() to Arena().

```
1 import greenfoot.*; // (World, Actor, GreenfootImage
2
3 /**
4  * Arena for tower defense game.
5  *
6  * @author Michal Varga
7  * @version 1.0
8  */
9 public class Arena extends World
10 {
11
12     /**
13     * Constructor for objects of class Arena.
14     *
15     */
16     public Arena()
17     {
18         // Create a new arena with 24x12 cells with a
19         super(24, 12, 50);
20     }
21 }
22
```

Figure 17. Rename class MyWord to Arena

Task: Create layout of Arena

Objective: Understanding the concept of constructors and relate them to the game scenario.

Concepts to Discuss: Constructors, Classes, Objects, Methods, Parameters, Keywords: *super, new, this*.

Activity: In this activity, a custom layout for Arena should be created. The custom layout should be provided within the constructor of the Arena class: one instance of Enemy, one instance of Orb, and at least one instance of Direction should be added. After declaring and initializing the variables, properties should be assigned by invoking the appropriate methods. Finally, these objects should be incorporated into the arena by invoking the **addObject(Actor)** method.

Commit: [8b105ea2eaf697f08c321efe687ddd31e2d0a041](https://github.com/8b105ea2eaf697f08c321efe687ddd31e2d0a041)

Solution: Custom layout for Arena is created within the constructor of the Arena class.


```
16 public Arena()  
17 {  
18     // Create a new arena with 24x12 cells  
19     super(24, 12, 50);  
20  
21     // DIRECTIONS  
22     Direction d1 = new Direction();  
23     d1.setRotation(270);  
24     this.addObject(d1, 8, 6);  
25     // --  
26     Direction d2 = new Direction();  
27     this.addObject(d2, 8, 1);  
28     // --  
29     Direction d3 = new Direction();  
30     d3.setRotation(90);  
31     this.addObject(d3, 16, 1);  
32     // --  
33     Direction d4 = new Direction();  
34     this.addObject(d4, 16, 10);  
35  
36     // ORB  
37     Orb o = new Orb();  
38     this.addObject(o, 23, 10);  
39  
40     // ENEMY  
41     Enemy e = new Enemy();
```

Figure 18. Create layout of Arena

6.3. Introduction to Attributes in the Greenfoot Environment

Title	Introduction to Attributes in the Greenfoot Environment
Learning objectives	By the end of this session, students will be able to understand the concept of attributes. The session introduces basic theoretical concepts related to attributes, as well as various code explanations and tasks. The examined concepts will be discussed in the context of the game development, encouraging creativity, teamwork, and an enthusiastic approach to coding in the Greenfoot environment.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge and basic object-oriented knowledge. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none"> 1) Task: Movement-related problem and solution (30 minutes) 2) Attributes (10 minutes) 3) Parameters of constructors (10 minutes) 4) Task: Enemy.moveDelay (20 minutes) 5) Task: Movement of enemies respecting delay (30 minutes)
Materials & resources	<p>The textbook from the OOP4Fun project.</p> <p>Resources from OOP4Fun project.</p> <p>Project source code from Github/Gitlab.</p> <p>Internet resources.</p>
Description	<p>During this 100-minute learning session, secondary school students are introduced to concepts related to attributes within the context of game development using the Greenfoot tool. The session starts with a 30-minute task related to movement-related problems and potential solutions.</p> <p>Based on the previous task, concepts related to attributes are introduced in the next 10-minute session. This is followed by a 10-minute scenario, during which parameters of constructors are explained and discussed.</p> <p>The next 20-minute teacher-guided session is related to task. New attribute related to movement in Enemy class is defined. In addition, the parametric constructor is defined. The feedback provided by the teacher and peers is included in this session.</p> <p>Finally, in the last 30-minute teacher-guided session movement of enemies is implemented. In this context, the act() method is updated. The feedback provided by the teacher and peers is included in this session.</p> <p>The students will continue working on the game project initiated in the previous sessions. Consequently, by the end of the session, they will be introduced to concepts related to attributes.</p>

Assessment	<p>The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p> <p>The state of the project opens possibilities for home assignments. In this context, more classes, expressions and values can be introduced to achieve additional behavior (e.g. teleports, tunnels, etc.). These concepts can be discussed with students and the respective implementation can be assigned as homework.</p>
Result dissemination	<p>In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.</p>

6.3.1. A teacher's Guide to Lesson Preparation

1. Task: Movement-related problem and solution

Objective: Understanding the movement-related problem and potential solutions in the context of constructors and attributes.

Concepts to Discuss: Constructors, Attributes, Methods.

Activity: The teacher should explain that the Enemy is currently moving two cells at once, which causes issues with its movement. To address this, the speed of the Enemy can be modeled differently. The Enemy instance will now always move one cell at a time. Additionally, a new attribute called **moveDelay** can be defined, which will cause the Enemy instance to move only after a certain number of **act()** method calls have passed.

2. Attributes

Objective: Understanding the concept of attributes.

Concepts to Discuss: Constructors, Classes, Objects, Attributes.

Activity: The teacher introduces the concept of attributes within the context of Class and Object concepts in Object-Oriented Programming (OOP).

3. Parameters of constructors

Objective: Understanding the concept of parameters of the constructor.

Concepts to Discuss: Constructors, Classes, Objects, Attributes, Parameters, Keywords: *super, new, this*.

Activity: The teacher introduces the concept of parameters of the constructor in context of Class and Object concepts in Object-Oriented Programming (OOP).

4. Task: Enemy.moveDelay

Objective: Understanding the concept of attributes and parameters of the constructor.

Concepts to Discuss: Constructors, Classes, Objects, Attributes, Parameters, Keywords: *super, new, this*.

Activity: A new attribute named **moveDelay** of type `int` will be added to the `Enemy` class. A parameterized constructor will also be defined to initialize this attribute, with the attribute being set to the value provided by the parameter. The code in the `Arena` class will be adjusted accordingly.

Commit: [6092489ce57541e77ae4e2ee886b20853df9f8a4](https://github.com/6092489ce57541e77ae4e2ee886b20853df9f8a4).

Solution: A new attribute `moveDelay` and parameterized constructor are added in the `Enemy` class.

```
9 public class Enemy extends Actor
10 {
11     private int moveDelay;
12
13     public Enemy(int moveDelay)
14     {
15         this.moveDelay = moveDelay;
16     }
17
18     /**
19      * Enemy moves two cells in its current
20      * If it steps on the Direction instance
21      */
22     public void act()
23     {
24         this.move(2);
25     }
26 }
```

Figure 19. `Enemy.moveDelay`

5. Task: Movement of enemies respecting delay

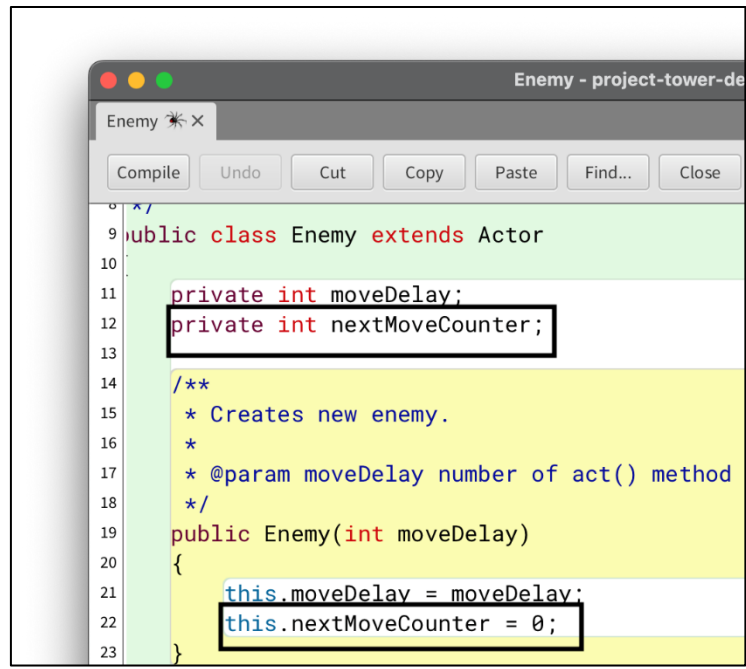
Objective: Understanding the concept of attributes and parameters of the constructor.

Concepts to Discuss: Constructors, Classes, Objects, Attributes, Parameters, Keywords: *super, new, this*.

Activity: The `act()` method of the `Enemy` class will be updated so that the `Enemy` moves only after the specified number of `moveDelay` calls of the method. Additionally, a new attribute called `nextMoveCounter` of type `int` will be introduced and initialized to 0. The `act()` method will be modified to call `this.move(1)` only when `nextMoveCounter` reaches 0. After the movement, `nextMoveCounter` will be reset to the value of `moveDelay`. If the `Enemy` instance cannot move because `nextMoveCounter` has not yet reached 0, `nextMoveCounter` will be decreased by 1.

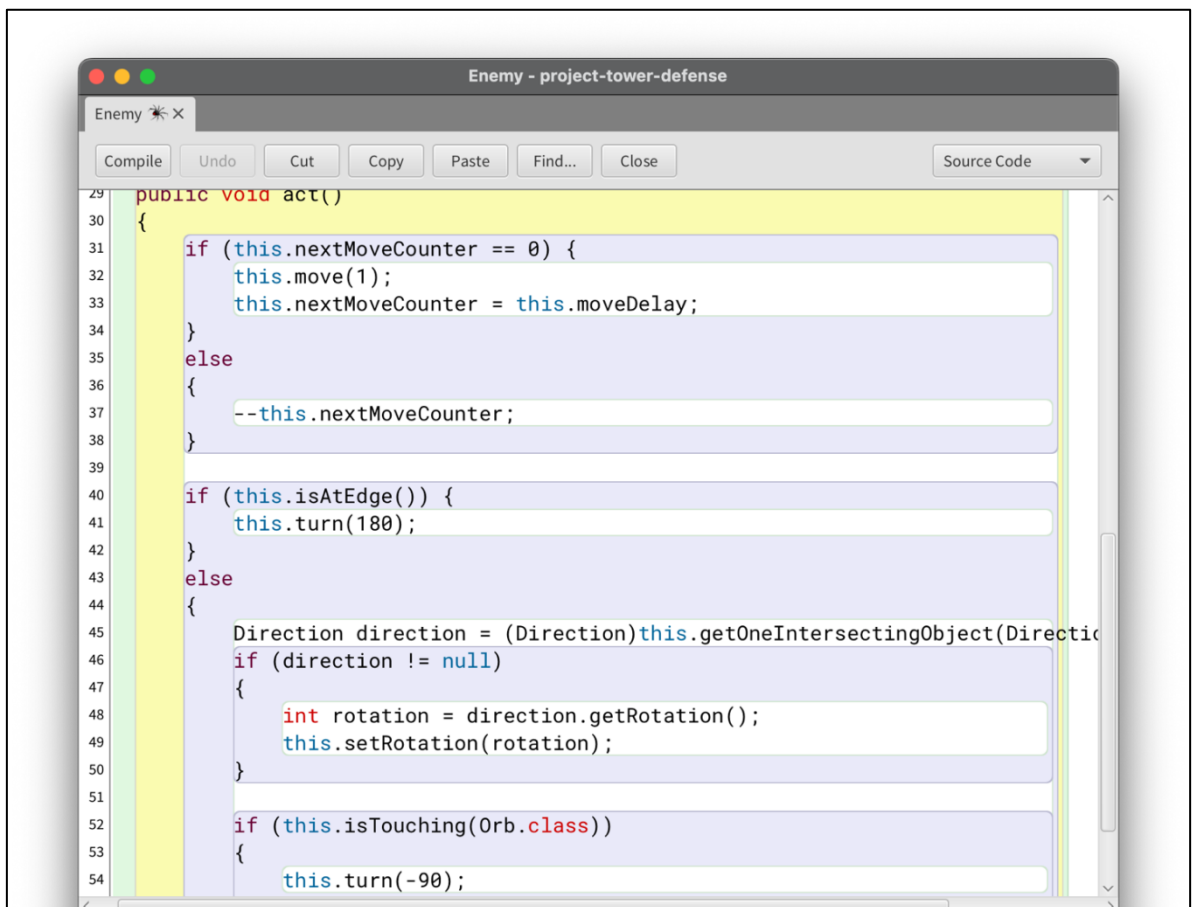
Commit: [bf26e6ed23911ccb712fae3e243cdedff3a89a7f](https://github.com/bf26e6ed23911ccb712fae3e243cdedff3a89a7f).

Solution: A new attribute `nextMoveCounter` was added and initialized in the constructor. Implementation of `act()` method was updated so that the `Enemy` moves only after the specified number of `moveDelay` calls of the method.



```
9 public class Enemy extends Actor
10
11     private int moveDelay;
12     private int nextMoveCounter;
13
14     /**
15      * Creates new enemy.
16      *
17      * @param moveDelay number of act() method
18      */
19     public Enemy(int moveDelay)
20     {
21         this.moveDelay = moveDelay;
22         this.nextMoveCounter = 0;
23     }
```

Figure 20- Movement of enemies respecting delay - 1



```
29 public void act()
30 {
31     if (this.nextMoveCounter == 0) {
32         this.move(1);
33         this.nextMoveCounter = this.moveDelay;
34     }
35     else
36     {
37         --this.nextMoveCounter;
38     }
39
40     if (this.isAtEdge()) {
41         this.turn(180);
42     }
43     else
44     {
45         Direction direction = (Direction)this.getOneIntersectingObject(Directio
46         if (direction != null)
47         {
48             int rotation = direction.getRotation();
49             this.setRotation(rotation);
50         }
51
52         if (this.isTouching(Orb.class))
53         {
54             this.turn(-90);
55         }
56     }
```

Figure 21 Movement of enemies respecting delay - 2

6.4. Introduction to Constructor Overloading in the Greenfoot Environment

Title	Introduction to Constructor Overloading in the Greenfoot Environment
Learning objectives	By the end of this session, students will be able to understand the concept of constructor overloading. The session introduces basic theoretical concepts related to constructor overloading, as well as various code explanations and tasks. The examined concepts will be discussed in the context of the game development, encouraging creativity, teamwork, and an enthusiastic approach to coding in the Greenfoot environment.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge and basic object-oriented knowledge. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none"> 1) Basic concepts of constructor overloading (5 minutes) 2) Task: Parametric constructor of class Direction (25 minutes) 3) Task: Overload constructors in class Direction (25 minutes) 4) Theory revision (20 minutes)
Materials & resources	<p>The textbook from the OOP4Fun project.</p> <p>Resources from OOP4Fun project.</p> <p>Project source code from Github/Gitlab.</p> <p>Internet resources.</p>
Description	<p>During this 75-minute learning session, secondary school students are introduced to concepts related to constructor overloading within the context of game development using the Greenfoot tool. The session starts with a 5-minute introduction to basic concepts of constructor overloading.</p> <p>The next 25-minute teacher-guided session is related to task in which parametric constructor in the class Direction is defined. In the next 25-minute task overloaded constructor in the class Direction is defined. The feedback provided by the teacher and peers is included in this sessions.</p> <p>Finally, in the last 20-minute teacher-guided session theory revision related to previously discussed concepts is performed (i.e., variables, expressions, operators, constructors, attributes, and constructors overloading).</p>
Assessment	<p>The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p> <p>The state of the project opens possibilities for home assignments. In this context, more classes, expressions and values can be introduced to achieve additional behavior (e.g. teleports, tunnels, etc.).These concepts can be discussed with students and the respective implementation can be assigned as homework.</p>
Result	In order to disseminate their results to teachers and fellow students the usual

dissemination	setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.
---------------	---

6.4.1. A teacher's Guide to Lesson Preparation

1. Basic concepts of constructor overloading

Objective: Understanding the concepts of constructors overloading.

Concepts to Discuss: Constructors.

Activity: The concepts of constructors overloading are discussed.

2. Task: Parametric constructor of class Direction

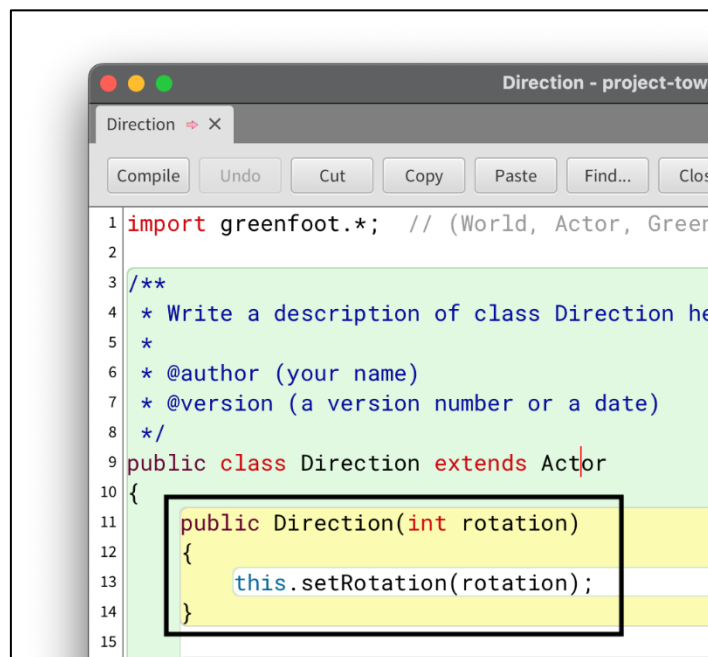
Objective: Definition of parametric constructor of Direction class in the context of game development.

Concepts to Discuss: Constructors, Parameters, Attributes, Parametric Constructors.

Activity: In this session, a parameterized constructor is defined for the Direction class with a single parameter, **rotation**, of type int. Within the constructor body, the created instance should be rotated based on the value of this parameter. The code in the Arena class should be updated accordingly.

Commit: [3c4b9ef57ab17bac2a0abc7fc5e76ea4b6e27e4b](https://github.com/3c4b9ef57ab17bac2a0abc7fc5e76ea4b6e27e4b).

Solution: Parameterized constructor in Direction class was defined. The code in the Arena class was updated accordingly.



The image shows a screenshot of an IDE window titled "Direction - project-towe". The code editor displays the following Java code:

```

1 import greenfoot.*; // (World, Actor, Green
2
3 /**
4  * Write a description of class Direction here
5  *
6  * @author (your name)
7  * @version (a version number or a date)
8  */
9 public class Direction extends Actor
10 {
11     public Direction(int rotation)
12     {
13         this.setRotation(rotation);
14     }
15

```

The constructor definition on lines 11-14 is highlighted with a yellow background and enclosed in a black rectangular box.

Figure 22. Parametric constructor of class Direction - 1

```
16 public Arena()
17 {
18     // Create a new arena with 24x12 cells with
19     super(24, 12, 50);
20
21     // DIRECTIONS
22     Direction d1 = new Direction(270);
23     this.addObject(d1, 8, 6);
24     // --
25     Direction d2 = new Direction(0);
26     this.addObject(d2, 8, 1);
27     // --
28     Direction d3 = new Direction(90);
29     this.addObject(d3, 16, 1);
30     // --
31     Direction d4 = new Direction(0);
32     this.addObject(d4, 16, 10);
33
34     // ORB
35     Orb o = new Orb();
36     this.addObject(o, 23, 10);
37
38     // ENEMY
39     Enemy e = new Enemy(2);
40     this.addObject(e, 0, 6);
41 }
```

Figure 23. Parametric constructor of class Direction - 2

3. Task: Overload constructors in class Direction

Objective: Definition of overload constructor of Direction class in the context of game development.

Concepts to Discuss: Constructors, Constructors Overload.

Activity: In this session an overloaded constructor is defined in the Direction class. A non-parameterized constructor is added, and within its body, the parameterized constructor is invoked with the argument rotation set to 0. The code in the Arena class should be updated accordingly, using the non-parameterized version of the Direction class constructor where possible.

Commit: [1e67e67523c66acea4e93363c9a3173302f424c8](https://github.com/1e67e67523c66acea4e93363c9a3173302f424c8).

Solution: Overloaded constructor in Direction class was defined. The code in the Arena class was updated accordingly.


```
5 *
6 * @author (your name)
7 * @version (a version number or a date)
8 */
9 public class Direction extends Actor
10 {
11     public Direction()
12     {
13         this(0);
14     }
15
16     public Direction(int rotation)
17     {
18         this.setRotation(rotation);
19     }
20 }
```

Figure 24. Task Overload constructors in class Direction - 1

```
17 {
18     // Create a new arena with 24x12 cells
19     super(24, 12, 50);
20
21     // DIRECTIONS
22     Direction d1 = new Direction(270);
23     this.addObject(d1, 8, 6);
24     //
25     Direction d2 = new Direction();
26     this.addObject(d2, 8, 1);
27     // --
28     Direction d3 = new Direction(90);
29     this.addObject(d3, 16, 1);
30     //
31     Direction d4 = new Direction();
32     this.addObject(d4, 16, 10);
33
34     // ORB
35     Orb o = new Orb();
36     this.addObject(o, 23, 10);
37
38     // ENEMY
39     Enemy e = new Enemy(2);
40     this.addObject(e, 0, 6);
41 }
```

Figure 25. Task Overload constructors in class Direction - 2

4. Theory revision

Objective: Theory revision related to previously discussed concepts.

Concepts to Discuss: Variables, Expressions, Operators, Constructors, Attributes, Constructors Overloading.

Activity: A review of the previously discussed concepts was conducted during this session.

7. Association

Four teaching scenarios have been created within the Variable and expressions thematic unit.

7.0. Greenfoot Objects on a Mission: Exploring Methods and Associations

Title	Greenfoot Objects on a Mission: Exploring Methods and Associations
Learning objectives	<p>By the end of the scenario, students should have a solid understanding of how objects can interact with each other. An instance of the class Enemy interacts with other objects, particularly the class Orb, within the Greenfoot environment. They should demonstrate proficiency in creating and invoking methods within Java classes, specifically implementing and testing the methods <code>Arena.respawn(Enemy)</code> and <code>Orb.hit(Enemy)</code>. Additionally, students should comprehend and manage class attributes effectively, including defining and utilizing the attributes <code>Enemy.attack</code> and <code>Orb.hp</code>. Students should be proficient in encapsulating data within a class, demonstrated by creating getters such as <code>getEnemy.attack()</code>, and understand the importance of data encapsulation for secure and maintainable code. They should understand and implement message passing between objects, ensuring that instances of classes communicate effectively to perform game actions. Furthermore, students should apply method calling techniques to solve interactive game development tasks, effectively utilizing syntax and parameters required for method invocation.</p>
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including iteration and selection concepts. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none">1) Task 5.1 - Discuss what should happen when enemy reaches orb(10 minutes)2) Task 5.2 - Discuss how instance of class Enemy should interact with the relevant objects using messages when hitting instance of class Orb (15 minutes)3) Task 5.3 - Attribute <code>Enemy.attack</code> and <code>Orb.hp</code> (10 minutes)4) Method (15 minutes)5) Task 5.4 - Getter of attribute <code>Enemy.attack</code> (5 minutes)6) Task 5.5 - Create and test method <code>Arena.respawn(Enemy)</code> (10 minutes)7) Task 5.6 - Create and test method <code>Orb.hit(Enemy)</code> (10 minutes)

<p>Materials & resources</p>	<p>The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.</p>
<p>Description</p>	<p>In this 75-minute learning scenario, secondary school students will dive into the object interactions, method creation, and attribute handling within the Greenfoot environment. This session aims to enhance students' understanding of how objects communicate and interact, a crucial aspect of object-oriented programming.</p> <p>The session begins with a 10-minute discussion on the dynamics between enemy objects and orbs, exploring what should happen when an enemy reaches an orb. This will set the stage for understanding object interactions in a game context.</p> <p>Following this, students will engage in a 15-minute task discussing how an instance of the class Enemy should interact with relevant objects using messages when it collides with an instance of the class Orb. This discussion will emphasize the importance of object communication and message passing.</p> <p>Next, a 10-minute task will focus on the attributes Enemy.attack and Orb.hp. Students will define and understand these attributes, crucial for managing the game's mechanics.</p> <p>In the subsequent 15 minutes, students will jump into methods, learning how to create and implement them within their classes. This section will solidify their understanding of method creation and invocation.</p> <p>In 5-minute task students will have to create a getter for the attribute Enemy.attack, reinforcing their knowledge of encapsulation and data retrieval.</p> <p>The next 10 minutes will be dedicated to creating and testing the method Arena.respawn(Enemy). Students will implement this method to handle the respawn logic for enemy objects, ensuring their understanding of method functionality and testing.</p> <p>Following this, another 10-minute task will involve creating and testing the method Orb.hit(Enemy), which will handle the interaction logic when an enemy hits an orb.</p> <p>Throughout the session, students will work individually or in small groups, promoting collaboration and peer learning. By actively participating in discussions, coding tasks, and testing methods, students will develop critical thinking skills and computational problem-solving abilities.</p> <p>At the conclusion of the session, students will emerge with a comprehensive</p>

	understanding of object interactions, method creation, and attribute management in Greenfoot. These essential skills will equip them for further game development projects and enhance their overall programming proficiency.
Assessment	The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.
Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.

7.0.1. A Teacher's Guide to Lesson Preparation

1) Task 5.1 - Discuss What Should Happen When Enemy Reaches Orb (10 minutes)

Objective: The teacher facilitates a discussion about the expected behavior when an enemy reaches the orb. Students are encouraged to think about the game dynamics and outcomes, such as the orb taking damage or the game ending.

Concepts to Discuss: Damage to the orb, enemy removal, triggering game events (e.g., reducing health, playing sound effects, ending the game).

Activity: The lesson begins with a review of previously covered concepts to ensure students have a solid foundation for the new material. The teacher engages students in a discussion to clarify the concepts of object interaction, messages, and methods within the context of the Greenfoot environment.

Students collaborate in groups to brainstorm the outcomes when an enemy reaches the orb in their game. They explore algorithms such as reducing the orb's health points (HP) upon enemy contact, discussing scenarios where the orb's HP might drop to zero, resulting in the game ending. Alternatively, if the orb's HP remains above zero, they agree to respawn the enemy in a different arena location. They also consider integrating additional game events triggered by this interaction, such as sound effects or on-screen messages. Throughout the session, the teacher guides the discussion, prompting students to align their proposed algorithms with a specific scenario: ensuring that when an enemy reaches the orb, the orb's HP decreases as previously outlined. This guidance helps students apply their ideas practically, reinforcing their understanding of game dynamics and interactions within their game's framework.

2) Task 5.2 - Discuss How Instance of Class Enemy Should Interact with Relevant Objects Using Messages When Hitting Instance of Class Orb (15 minutes)

Objective: The teacher guides students through the process of understanding how an instance of the Enemy class should interact with other objects, specifically when it hits an instance of the Orb class, using messages.

Concepts to Discuss: Method calling, parameter passing, object references.

Activity: Students collaborate in pairs to explore how an instance of the Enemy class should interact with an instance of the Orb class using messages within their game scenario. They analyze and map out the sequence of messages and actions that should unfold when an enemy hits the orb. This exercise aims to deepen their understanding of method calling, parameter passing, and object references in the context of game development. As students discuss and refine their ideas, the teacher facilitates the session, guiding them to ensure that the sequence of interactions among the objects follows the algorithm spread across cooperating objects, as specified in the lesson's objectives. To aid in this process, the teacher can introduce and utilize a UML sequence diagram to visually describe the interactions among the Enemy, Orb, Arena, and Greenfoot classes. This visual tool helps students better comprehend the flow of messages and method calls, reinforcing their understanding of object-oriented programming concepts and their application in Java programming within the Greenfoot environment.

3) Task 5.3 - Attribute Enemy.attack and Orb.hp (10 minutes)

Objective: The teacher introduces the attributes Enemy.attack and Orb.hp, explaining their significance in determining the outcome of interactions.

Concepts to Discuss: Class attributes, encapsulation.

Activity: The teacher introduces the concept of class attributes and encapsulation, explaining how attributes like Enemy.attack and Orb.hp can represent essential characteristics of objects within the game. Students learn to define and use these attributes in their code to model the enemy's attack power and the orb's health points. They start by adding a new integer attribute called attack to the Enemy class, including a parameter in the constructor to initialize this attribute. Similarly, they add an integer attribute called hp to the Orb class, along with a parametric constructor to set this value upon object creation. The teacher guides students through adjusting the code in the Arena class to accommodate these new attributes. This hands-on experience helps students understand the role of class attributes in object interactions and the importance of encapsulation in maintaining code integrity and security.

Commit: [4ca1e9f25685990d2bdfe5b610c28422e0944f95](https://github.com/4ca1e9f25685990d2bdfe5b610c28422e0944f95)

Solution: The solution requires changes to three classes: Enemy, Orb and Arena.

For start, it is necessary to modify the Enemy class by defining a new attribute and initializing it using a parameter in the constructor.

```

12 private int nextMoveCounter;
13 private int attack;
14
15 /**
16  * Creates new enemy.
17  *
18  * @param moveDelay number of act() method calls to make a move.
19  */
20 public Enemy(int moveDelay, int attack)
21 {
22     this.moveDelay = moveDelay;
23     this.nextMoveCounter = 0;
24     this.attack = attack;
25 }
26

```

Figure 26. Task 5.3 - 1

A similar modification needs to be made in the Orb class.

```

9 public class Orb extends Actor
10 {
11     private int hp;
12
13     public Orb(int hp)
14     {
15         this.hp = hp;
16     }
17

```

Figure 27. Task 5.3 - 2

Finally, it is necessary to modify the Arena class to use the constructors appropriately and initialize the attribute values.

```

34 // ORB
35 Orb o = new Orb(10);
36 this.addObject(o, 23, 10);
37
38 // ENEMY
39 Enemy e = new Enemy(2, 1);
40 this.addObject(e, 0, 6);

```

Figure 28. Task 5.3 - 3

4) Method (15 minutes)

Objective: The teacher provides an overview of methods, explaining how they are used to encapsulate actions and behaviors within classes.

Concepts to Discuss: Method definition, method invocation, parameters, return values.

Activity: The teacher begins by explaining the concept of methods as encapsulated actions or behaviors within a class. Using practical examples, the teacher demonstrates the syntax and structure of method definitions, illustrating how methods are invoked on objects. Students learn about different types of methods, including those that perform actions (void methods) and those that return values (return type methods). The teacher explains how parameters are passed to methods, highlighting the importance of parameter types and order. Through guided coding exercises, students practice defining methods with various parameter and return types, and invoking these methods on object instances. They explore scenarios where methods perform actions, modify object states, or return specific values, solidifying their understanding of method functionality within a class.

5) Task 5.4 - Getter of Attribute Enemy.attack (5 minutes)

Objective: The teacher explains the concept of getter methods and their purpose in accessing attribute values.

Concepts to Discuss: Accessor methods (getters), encapsulation.

Activity: The teacher begins by explaining the purpose of getter methods, emphasizing how they provide controlled access to attribute values while maintaining encapsulation. Students learn the importance of using getters to retrieve private attribute values, reinforcing the concept of data protection within a class. The teacher then guides students through the process of creating a getter method for the attack attribute in the Enemy class. Using a hands-on approach, students implement the getter method, ensuring it returns the value of the attack attribute. The teacher demonstrates the correct syntax and structure for defining a getter, and how it is used within the code to access the attribute value. By the end of the activity, students should be able to create and utilize getter methods to access attribute values in a controlled manner, enhancing their understanding of encapsulation and data protection in object-oriented programming.

Commit: [72b7456ea4cc11416c57d72c89b6a7f7e9266e3e](https://github.com/72b7456ea4cc11416c57d72c89b6a7f7e9266e3e)

Solution: The solution requires changes to the class Enemy.

In the Enemy class, a getter method for the attack attribute should be added.

```
26 |
27 | public int getAttack()
28 | {
29 |     return this.attack;
30 | }
31 |
```

Figure 29. Task 5.4

6) Task 5.5 - Create and Test Method `Arena.respawn(Enemy)` (10 minutes)

Objective: The teacher guides students through the creation and testing of the `Arena.respawn(Enemy)` method, which handles respawning enemies in the game.

Concepts to Discuss: Method implementation, testing, game mechanics.

Activity: The teacher begins by introducing the concept of method implementation and its importance in defining specific behaviors within a class. Emphasizing practical application, students are guided to create the `respawn` method in the `Arena` class. This method, which does not return a value, takes a single parameter of type `Enemy`. Students are instructed to set the location and rotation of the enemy within this method to match the values initially set in the constructor. The teacher demonstrates the correct syntax and structure for defining this method, reinforcing key concepts of method implementation and parameter passing.

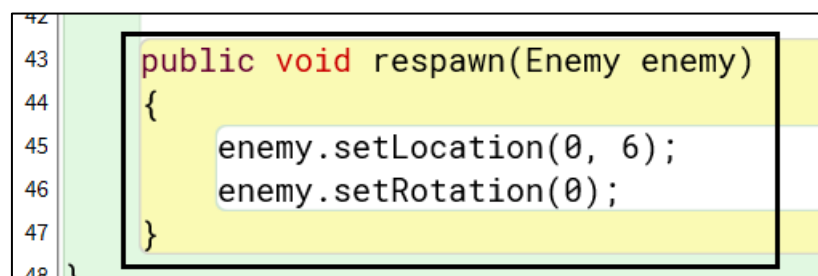
Next, students test their method to ensure it functions correctly. They create an instance of the `Arena` and an `Enemy` but do not launch the application immediately. Instead, they drag the `Enemy` instance to a new location, then access the context menu of the `Arena` instance to invoke the `respawn` method. The teacher explains the process of ensuring the application is paused and the parameter field is active, guiding students to right-click on the `Enemy` instance to fill the parameter field correctly. Students observe the expression built in the window and then click the `OK` button to see the effect of their `respawn` method.

Through this activity, students gain practical experience in writing and testing methods, understanding how to manipulate game objects programmatically. The teacher ensures that students comprehend each step, providing assistance and clarification as needed, thereby reinforcing their understanding of method implementation and game mechanics.

Commit: [43a221876b8acb4fd507175ec4c8f520121d1ab1](https://github.com/43a221876b8acb4fd507175ec4c8f520121d1ab1)

Solution: The solution requires changes to the class `Arena`.

In the `Arena` class, a `respawn()` method should be defined.



```
42
43 public void respawn(Enemy enemy)
44 {
45     enemy.setLocation(0, 6);
46     enemy.setRotation(0);
47 }
48 }
```

Figure 30. Task 5.5

7) Task 5.6 - Create and Test Method `Orb.hit(Enemy)` (10 minutes)

Objective: The teacher instructs students on creating and testing the `Orb.hit(Enemy)` method, which defines the interaction when an enemy hits the orb.

Concepts to Discuss: Method interaction, updating object state.

Activity:The teacher begins by explaining the purpose of the `Orb.hit(Enemy)` method, emphasizing how it encapsulates the interaction logic between the orb and an enemy. Students are then guided to add this method to the `Orb` class. This method, which does not return a value, takes a single parameter of type `Enemy`.

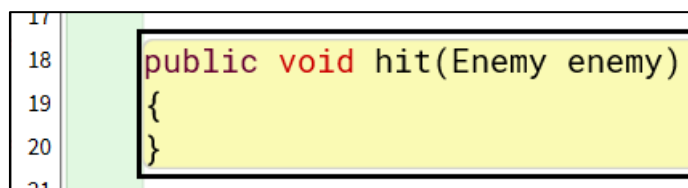
To test the method, students follow a step-by-step process similar to the one used for testing the `respawn` method. They create an instance of the `Orb` class and an instance of the `Enemy` class. Without launching the application, they invoke the context menu of the `Orb` instance and select the `hit` method. The teacher ensures students understand how to fill the parameter field by right-clicking on the `Enemy` instance while the application is paused. This action builds the method call expression, which students then execute by clicking the `OK` button.

The teacher emphasizes the importance of observing the expression built in the window to verify the method call. This exercise helps students understand method interaction and the process of updating object states within a game context. Through this hands-on activity, students gain practical experience in implementing and testing methods, reinforcing their comprehension of method interaction and game behavior. The teacher provides support and clarification as needed, ensuring students successfully complete the task and understand its significance.

Commit: [fe03d520260f172066be35055a901487bf7c2ff7](https://github.com/fe03d520260f172066be35055a901487bf7c2ff7)

Solution:The solution requires changes to the class `Orb`.

In the `Orb` class, a `hit()` method should be defined.

A screenshot of a code editor showing the implementation of the `hit` method in the `Orb` class. The code is displayed on lines 18 through 21. The method signature is `public void hit(Enemy enemy)`, followed by an opening curly brace on line 19 and a closing curly brace on line 20. The code is highlighted in yellow.

```
17  
18 public void hit(Enemy enemy)  
19 {  
20 }  
21
```

Figure 31. Task 5.6

7.1. Greenfoot Objects on a Mission: Exploring Associations and Advanced Method Calls

Title	Greenfoot Objects on a Mission: Exploring Associations and Advanced Method Calls
Learning objectives	<p>By the end of the scenario, students should develop a deep understanding of how objects in different classes can form associations and interact effectively within the Greenfoot environment. Students should demonstrate proficiency in calling the <code>Orb.hit(Enemy)</code> method from the <code>Enemy</code> class, showcasing their ability to initiate advanced method calls and facilitate communication between objects. They should be able to explain the functionality of key Greenfoot methods such as <code>Greenfoot.stop()</code> and <code>World.getWorldOfType(_cls_)</code>, understanding their roles in controlling game execution and managing object instances effectively, and also, students should successfully implement the <code>Orb.hit(Enemy)</code> method within their projects, integrating object interactions and method functionalities to create interactive and dynamic game mechanics.</p> <p>Students should apply fundamental principles of object-oriented programming, including encapsulation and method invocation, to develop sophisticated game interactions and functionalities. They should gain practical insights into various aspects of game development, including enemy spawning, game state management, and the creation of engaging player experiences through structured object interactions.</p>
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including iteration and selection concepts. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none"> 1) Association (10 minutes) 2) Task 5.7 - Call method <code>Orb.hit(Enemy)</code> from <code>Enemy</code> (15 minutes) 3) Explanation of the code for methods <code>Greenfoot.stop()</code> and <code>World.getWorldOfType(_cls_)</code> (15 minutes) 4) Task 5.8 - Implement method <code>Orb.hit(Enemy)</code>(30 minutes)
Materials & resources	<p>The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.</p>

Description	<p>In this 70-minute learning scenario, secondary school students will delve into the details of associations between objects and advanced method calls within the Greenfoot environment. The session aims to deepen students' understanding of object-oriented programming concepts and enhance their ability to implement complex interactions in game development.</p> <p>The session begins with a 10-minute discussion on associations between classes, focusing on how objects can interact and collaborate within a software system. This foundational understanding sets the stage for exploring more complex interactions.</p> <p>Following this, students will engage in a 15-minute task aimed at calling the method <code>Orb.hit(Enemy)</code> from the <code>Enemy</code> class. This task emphasizes the practical application of method invocation and message passing between objects.</p> <p>The next segment involves a detailed explanation of the code for methods <code>Greenfoot.stop()</code> and <code>World.getWorldOfType(_cls_)</code>, taking 15 minutes. Students will gain insights into how these methods function within the Greenfoot framework, enabling precise control over game elements and world management.</p> <p>The core of the session is dedicated to a 30-minute task where students will implement the method <code>Orb.hit(Enemy)</code>. This task challenges students to apply their understanding of method implementation, parameter passing, and object interactions to create a functional game mechanic within their projects.</p> <p>At the conclusion of the scenario, students will emerge with a deeper understanding of associations between objects, proficiency in advanced method calls like <code>Orb.hit(Enemy)</code> from the <code>Enemy</code> class, and insights into the implementation of crucial Greenfoot methods. These skills will equip them to create more interactive and dynamic games, leveraging the full potential of object-oriented programming principles in game development.</p>
Assessment	<p>The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p>
Result dissemination	<p>In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.</p>

7.1.1. A Teacher's Guide to Lesson Preparation

1) Association (10 minutes)

Objective: The teacher explains the concept of association between objects, emphasizing how objects of different classes can interact with each other.

Concepts to Discuss: Associations, object interactions, class relationships.

Activity: The lesson begins with a brief review of associations between classes in object-oriented programming. The teacher engages students in a discussion to clarify how objects interact with each other through associations, using practical examples from the Greenfoot environment to illustrate these concepts. Students delve into understanding that associations define how classes collaborate, such as how an Enemy can affect an Orb in a game scenario.

Through a detailed discussion, the teacher elaborates on the different types of associations within Greenfoot: one-to-one, one-to-many, and many-to-many. For instance, a one-to-one association might illustrate how a Player is linked to their Character avatar, a one-to-many association could show how a World contains multiple Actor instances, and a many-to-many association might depict how various Enemies interact with multiple Orbs in a game level.

Using UML class diagrams specific to Greenfoot, the teacher visually demonstrates these relationships, aiding students in understanding how associations are structured and implemented in their game projects. For example, the diagram could depict how an Enemy class is associated with multiple instances of the Orb class, indicating a one-to-many relationship where each Enemy affects several Orbs.

Students actively participate in identifying and mapping out these associations within their game projects. They explore how objects interact based on these relationships and discuss the implications for game mechanics and logic. The teacher provides concrete examples from their game development context, illustrating how Enemy instances interact with Orb instances and how these interactions are governed by associations.

By the end of the activity, students gain a solid grasp of the role associations play in designing and implementing interactive systems within Greenfoot. They can identify different types of associations and apply this knowledge to model and implement complex interactions between Greenfoot objects effectively. This practical understanding strengthens their ability to design cohesive and interactive game scenarios using object-oriented principles.

2) Task 5.7 - Call Method `Orb.hit(Enemy)` from Enemy (15 minutes)

Objective: The teacher guides students through the process of calling the `Orb.hit(Enemy)` method from within the Enemy class.

Concepts to Discuss: Method invocation, object references.

Activity: Students engage in hands-on learning by altering the `act()` method of the `Enemy` class. They remove existing code responsible for unnecessary behaviors like rotating upon hitting the orb or bouncing off the world's edges. Instead, they implement the functionality to call the `Orb.hit(Enemy)` method when an instance of `Enemy` collides with an instance of `Orb`.

Through practical examples and demonstrations, the teacher illustrates how to set up the method call correctly. Students learn to use object references (`this`) to trigger the `hit()` method on the `Orb` instance upon collision with an `Enemy`. They explore the flow of control in Greenfoot, understanding how method invocation directs the execution path within their game scenario.

Throughout the activity, the teacher provides guidance on debugging and testing the implementation to ensure that calling `Orb.hit(Enemy)` functions as intended. Students observe the behavior in the Greenfoot simulation environment, validating that the method call effectively triggers the expected interactions between `Enemy` and `Orb` objects.

By the end of the lesson unit, students gain proficiency in method invocation and object interaction within Greenfoot programming. They comprehend how to utilize object references to invoke methods across different classes, reinforcing their understanding of object-oriented programming principles in a game development context.

Commit: [63f9c96717d9d2587b60095e3b249b0158c8587b](https://github.com/63f9c96717d9d2587b60095e3b249b0158c8587b)

Solution: The solution requires changes to the class `Enemy`.

In the `act()` method of the `Enemy` class, the method `hit()` should be called.

```
37 public void act()
38 {
39     if (this.nextMoveCounter == 0)
40     {
41         this.move(1);
42         this.nextMoveCounter = this.moveDelay;
43     }
44     else
45     {
46         --this.nextMoveCounter;
47     }
48
49     Direction direction = (Direction) this.getOneIntersectingObject(Direction.class);
50     if (direction != null)
51     {
52         int rotation = direction.getRotation();
53         this.setRotation(rotation);
54     }
55
56     Orb orb = (Orb) this.getOneIntersectingObject(Orb.class);
57     if (orb != null)
58     {
59         orb.hit(this);
60     }
61 }
```

Figure 32. Task 5.7

3) Explanation of the Code for Methods `Greenfoot.stop()` and `World.getWorldOfType(cls)` (15 minutes)

Objective: The teacher explains the functionality of the `Greenfoot.stop()` method and the `World.getWorldOfType(_cls_)` method.

Concepts to Discuss: Game control methods, world management.

Activity: In this lesson unit, students delve into understanding two crucial methods within the Greenfoot framework: `Greenfoot.stop()` and `World.getWorldOfType(_cls_)`. The teacher begins by elucidating the purpose and usage of each method, emphasizing their roles in game control and world management.

The `Greenfoot.stop()` method is essential for controlling the execution flow of a Greenfoot scenario. When invoked, it halts the simulation and freezes all actors and interactions within the world. This method is particularly useful for implementing game pause functionality or triggering specific events that require stopping the game's progression temporarily.

On the other hand, the `World.getWorldOfType(_cls_)` method serves a different purpose related to world management. This method allows developers to retrieve instances of worlds that are of a specific class type (`_cls_`). It traverses through all active worlds in the Greenfoot environment and returns instances of the world class that match the specified type. This capability is beneficial when developers need to interact with or manipulate worlds dynamically based on their class attributes.

Students engage in practical demonstrations and examples to deepen their understanding of these methods. The teacher showcases how `Greenfoot.stop()` can be integrated into game scenarios to create pause functionality or trigger specific in-game events. Students observe how pausing the game affects actor behaviors and interactions within the Greenfoot simulation environment.

Similarly, students explore the `World.getWorldOfType(_cls_)` method through hands-on exercises. They learn how to use this method to retrieve instances of specific world types dynamically. The teacher demonstrates scenarios where fetching worlds of a particular class type is necessary for implementing advanced game mechanics or managing multiple concurrent game environments within Greenfoot.

Throughout the activity, the teacher encourages discussion and provides practical coding examples to illustrate the applications of these methods in real-world game development scenarios. Students actively participate in experimenting with the methods within their own Greenfoot projects, reinforcing their comprehension through direct application and exploration.

By the end of the lesson unit, students gain proficiency in utilizing `Greenfoot.stop()` for game control and `World.getWorldOfType(_cls_)` for efficient world management within the Greenfoot environment. They acquire practical skills that enhance their ability to

implement complex game behaviors and manage game states effectively using these essential methods.

4) Task 5.8 - Implement Method `Orb.hit(Enemy)` (30 minutes)

Objective: The teacher guides students through the implementation of the `Orb.hit(Enemy)` method.

Concepts to Discuss: Method implementation, updating object state, game mechanics.

Activity: Students starts implementing the `Orb.hit(Enemy)` method, a crucial step in defining the interaction between an enemy and the orb within their game scenario.

The `Orb.hit(Enemy)` method plays a pivotal role in determining the consequences when an enemy collides with the orb in the game. Here's how students can approach and implement this method. First, decrease Orb's Health Points. Upon calling `Orb.hit(Enemy)`, the method should reduce the orb's health points (hp). This action simulates damage taken by the orb upon impact with an enemy. After that, check if the orb's health points have dropped to zero. If the orb's health points (hp) reach zero or below, the game should end. This is achieved by invoking `Greenfoot.stop()` to halt the game's execution. That is the game over scenario. Different scenario happens if the orb's health points are still above zero after the enemy collision, students should incorporate logic to respawn the enemy back into the arena for continuous gameplay.

During the activity, the teacher facilitates the implementation of `Orb.hit(Enemy)` by providing guidance on method structure, parameter usage (`Enemy`), and how to update the orb's state (hp). Students collaborate to discuss and decide on the specific game mechanics they want to implement, such as how much damage each enemy type inflicts on the orb and what happens when the orb's health points are depleted.

The teacher encourages students to test their implementations thoroughly, ensuring that the method behaves as expected in various game scenarios. By the end of the activity, students gain practical experience in implementing method logic to manage game interactions effectively, reinforcing their understanding of method implementation and game mechanics within the Greenfoot framework.

(Commit: [84bcd7c128faaa9313b507f7438f826ae2f47d2c](#))

Solution:The solution requires changes to the class `Orb`.

In the `hit()` method of the `Orb` class should be implemented.

```
17  
18 public void hit(Enemy enemy)  
19 {  
20     this.hp -= enemy.getAttack();  
21  
22     Arena arena = this.getWorldOfType(Arena.class);  
23     if (this.hp <= 0)  
24     {  
25         Greenfoot.stop();  
26     }  
27     else  
28     {  
29         arena.respawn(enemy);  
30     }  
31 }  
32
```

Figure 33. Task 5.8

7.2. Greenfoot Objects on a Mission: Towers, Bullets, and Strategic Interactions

Title	Greenfoot Objects on a Mission: Towers, Bullets, and Strategic Interactions
Learning objectives	<p>By the end of the scenario, students will develop proficiency in creating the Bullet and Tower classes, establishing a foundation for strategic game development. They will understand and implement realistic Bullet movement and determine actions when Bullet instances encounter Enemy instances or the arena's edge. Students will design effective shooting mechanics for Tower instances, utilizing message passing between Tower objects and other game elements to enhance gameplay dynamics. They will strategically deploy Tower instances within the game arena and apply object-oriented principles such as encapsulation and method invocation to ensure the creation of robust and maintainable game mechanics. Through collaborative problem-solving, students will address challenges in designing and implementing strategic interactions between towers, bullets, and game elements, gaining practical insights into game design principles and enhancing their overall understanding of tower defense mechanics in Greenfoot.</p>
Target audience	<p>Secondary school students attending the OOP4Fun course. Basic programming knowledge including iteration and selection concepts. Students should be introduced to Greenfoot.</p>
Scenario duration	<ol style="list-style-type: none"> 1) Task 5.9 - Create classes Bullet and Tower(10 minutes) 2) Task 5.10 - Discuss how the instance of class Bullet should move and what should happen when it reaches instance of class Enemy or edge of the arena. (10 minutes) 3) Task 5.11 - Implement movement of instance of class Bullet (30 minutes) 4) Task 5.12 - Discuss how the instance of class Tower will shoot instance of class Bullet (15 minutes) 5) Task 5.13 - Discuss how instance of class Tower should interact with the relevant objects using messages when shooting (15 minutes) 6) Task 5.14 - Implement shooting of instance of class Tower (30 minutes) 7) Task 5.15 - Towers in Arena (20 minutes)
Materials resources	<p>& The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.</p>

Description	<p>In this 130-minute learning scenario, secondary school students will immerse themselves in the dynamics of tower and bullet interactions within the Greenfoot environment. The session focuses on developing students' skills in designing and implementing strategic game elements to create engaging and interactive gameplay.</p> <p>The session commences with a 10-minute task where students create the Bullet and Tower classes. This foundational step sets the groundwork for understanding and implementing the interactions between these game elements.</p> <p>Following this, a 10-minute discussion ensues on how instances of the Bullet class should move and the actions that should occur when a bullet reaches an instance of the Enemy class or the edge of the arena. This discussion sets the stage for implementing precise and dynamic movement mechanics.</p> <p>Students then dedicate 30 minutes to implementing the movement of instances of the Bullet class. This task challenges students to apply their understanding of Greenfoot's <code>move()</code> method and event handling to simulate realistic bullet behavior within the game environment.</p> <p>The session continues with a 15-minute task discussing how instances of the Tower class should shoot instances of the Bullet class. This discussion covers the logic and conditions for initiating bullet shots from towers.</p> <p>Following this, another 15 minutes are dedicated to discussing how instances of the Tower class should interact with relevant objects using messages when shooting. This segment emphasizes the importance of object communication and event triggering in game development.</p> <p>Students then spend 30 minutes implementing the shooting mechanism of instances of the Tower class. This task requires students to integrate shooting logic with object interactions, ensuring that towers effectively engage with enemies or other game elements.</p> <p>The session concludes with a 20-minute task focusing on managing and deploying towers within the arena. This task explores the placement, interaction, and strategic positioning of towers to optimize gameplay dynamics and challenge players effectively.</p> <p>At the end of the session, students will have gained practical experience in designing and implementing strategic interactions between towers and bullets in Greenfoot. They will have acquired skills in object-oriented programming, event handling, and strategic game design, preparing them to create engaging and dynamic games.</p>
-------------	---

Assessment	The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.
Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.

7.2.1. A Teacher's Guide to Lesson Preparation

1. Task 5.9 - Create Classes Bullet and Tower (10 minutes)

Objective: The teacher guides students through the creation of the Bullet and Tower classes.

Concepts to Discuss: Class creation, class roles, and initial setup.

Activity: The lesson starts with a review of the core concepts related to object creation, movement, and interaction within the Greenfoot environment. The teacher engages students in a discussion to clarify the roles of different classes and their interactions in the game.

Students create two new classes in the Greenfoot environment, understanding the purpose of each in the context of the game. They learn how to set up these classes, preparing for more complex interactions in later lessons.

Students start by creating the Bullet class. This class will represent projectiles fired by the towers. They open Greenfoot, select "New Class" from the menu, and name the new class Bullet. ~~Initially, they define a basic structure for the class with act() method for the bullet's behavior.~~

Next, students create the Tower class. This class will represent the towers that shoot bullets at enemies. They again select "New Class" from the menu and name the new class Tower.

Throughout the session, the teacher explains the roles of the Bullet and Tower classes within the game. The Bullet class represents projectiles that the towers will fire, while the Tower class represents stationary objects that can shoot bullets at enemies. The teacher ensures students understand the distinct roles each class plays and how they will interact in the game.

By the end of this activity, students should have created and set up basic structures for the Bullet and Tower classes, laying the groundwork for more detailed implementation in future lessons.

(Commit: [ece4df70042c8f60098e14ad2cee55514897d825](https://github.com/yourusername/yourrepository/commit/ece4df70042c8f60098e14ad2cee55514897d825))

Solution:The solution requires creation of two new classes:Bullet and Tower.

For start, it is necessary to create the Bullet class.

```
1 import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
2
3 /**
4  * Write a description of class Bullet here.
5  *
6  * @author (your name)
7  * @version (a version number or a date)
8  */
9 public class Bullet extends Actor
10 {
11     /**
12     * Act - do whatever the Bullet wants to do. This method is called whenever
13     * the 'Act' or 'Run' button gets pressed in the environment.
14     */
15     public void act()
16     {
17         // Add your action code here.
18     }
19 }
20
```

Figure 34. Task 5.9 - 1

Next, the class Tower should be created.

```
1 import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
2
3 /**
4  * Write a description of class Tower here.
5  *
6  * @author (your name)
7  * @version (a version number or a date)
8  */
9 public class Tower extends Actor
10 {
11     /**
12     * Act - do whatever the Tower wants to do. This method is called whenever
13     * the 'Act' or 'Run' button gets pressed in the environment.
14     */
15     public void act()
16     {
17         // Add your action code here.
18     }
19 }
20
```

Figure 35. Task 5.9 - 2

2. Task 5.10 - Discuss How the Instance of Class Bullet Should Move and What Should Happen When It Reaches Instance of Class Enemy or Edge of the Arena (10 minutes)

Objective: The teacher leads a discussion on the expected behavior of a bullet as it moves in the game.

Concepts to Discuss: Movement logic, collision detection.

Activity: Students brainstorm and discuss the logic for bullet movement and collision handling.

Students start by discussing how the Bullet instance should move within the game. They agree that the bullet should move in a straight line in the direction it was fired, without changing direction. The speed of the bullet should be manageable and consistent. To implement this, they can use Greenfoot's built-in movement methods. It is important to highlight the role of the constructor for initializing attribute values when creating object instances.

The teacher then guides the discussion towards what should happen when the bullet reaches the edge of the world or collides with an enemy. The students propose that when a bullet reaches the edge of the world, it should be removed from the game. They also discuss that upon colliding with an enemy, the bullet should disappear, and the enemy should take damage or be destroyed.

Students suggest the following pseudocode for the bullet's movement and collision logic: first move the bullet forward at a constant speed, then check if the bullet has reached the edge of the world. If true, remove the bullet from the world. If not, check if the bullet has collided with an enemy. If there is a collision, that means that bullet hit the enemy, and we need to remove the bullet from the world and apply damage to the enemy or remove the enemy.

The teacher demonstrates how to implement this logic using the Greenfoot methods `move(int)`, `isAtEdge()`, and `getOneIntersectingObject(Class cls)`.

The teacher encourages students to refine their ideas and think about additional details, such as adjusting the speed based on game difficulty or adding visual effects when a bullet hits an enemy. This activity helps students understand the principles of movement and collision detection in game development, preparing them for further implementation in their projects.

3. Task 5.11 - Implement Movement of Instance of Class Bullet (30 minutes)

Objective: The teacher helps students implement the movement logic for the Bullet class.

Concepts to Discuss: Actor movement, world boundaries.

Activity: Students write code to move the bullet forward and handle its removal when it reaches the edge of the arena. The students start by reviewing the concepts of actor movement and world boundaries, focusing on how these concepts apply to the Bullet class. The teacher reminds students of previous tasks, emphasizing the knowledge they should

apply. Students recall how to add code to the act() method to handle interactions at the edge of the world. Also, they recall how to handle directional changes when an actor enters specific cells. Moreover, they remember how to use counters and delay mechanisms in the act() method for controlled movement.

With these concepts in mind, students proceed to implement the movement logic for the Bullet class. They start by adding the move(int) method in the act() method of the Bullet class to move the bullet forward continuously.

Commit: [d372827a831381b2254f838041fa4d9a42e53b82](https://github.com/d372827a831381b2254f838041fa4d9a42e53b82)

Solution:The solution requires changes to the classBullet.

For start, it is necessary to define new attributes, and proper constructor.

```
9 public class Bullet extends Actor
10 {
11     private int moveDelay;
12     private int nextMoveCounter;
13
14     public Bullet(int moveDelay)
15     {
16         this.moveDelay = moveDelay;
17         this.nextMoveCounter = 0;
18     }
19 }
```

Figure 36. Task 5.11 - 1

Next, the method act() should be implemented.

```
24 public void act()
25 {
26     if (this.nextMoveCounter == 0)
27     {
28         this.move(1);
29         this.nextMoveCounter = this.moveDelay;
30     }
31     else
32     {
33         --this.nextMoveCounter;
34     }
35
36     boolean enemyHit = false;
37
38     Enemy enemy = (Enemy) this.getOneIntersectingObject(Enemy.class);
39     if (enemy != null)
40     {
41         // hit enemy
42         enemyHit = true;
43     }
44
45     if (enemyHit || this.isAtEdge())
46     {
47         Arena arena = this.getWorldOfType(Arena.class);
48         arena.removeObject(this);
49     }
50 }
```

Figure 37. Task 5.11 - 2

4. Task 5.12 - Discuss How the Instance of Class Tower Will Shoot Instance of Class Bullet (15 minutes)

Objective: The teacher discusses with students the logic for shooting bullets from the tower.

Concepts to Discuss: Object creation, method invocation.

Activity: Students brainstorm how the tower will create and launch bullets. Students start by discussing the overall logic needed for a tower to shoot bullets in the game. They focus on key concepts such as creating bullet objects and invoking methods to launch them. The teacher highlights that the tower should not shoot bullets on every call of the act() method, similar to how enemy movement was handled with a delay mechanism.

Students brainstorm the steps required for the tower to shoot bullets intermittently. The teacher guides them to consider using a delay mechanism for shooting. It should be explained what the role of constructor for could be implementing this mechanism.

The teacher explains that they will need to introduce a new attribute, shootDelay, and a counter, nextShootCounter, in the Tower class. These attributes will control the shooting frequency.

The teacher breaks down the relevant steps and methods for the Tower class. First define the shootDelay attribute and initialize the nextShootCounter to 0 in the Tower class. After that, the act() method of the Tower class should be modified to handle the shooting logic. The method should only create and shoot a bullet when nextShootCounter reaches 0. After shooting, nextShootCounter should be reset to the value of shootDelay. If nextShootCounter is not 0, it should be decremented by 1. At the end, separate fire() method should be defined to handle the creation and launching of bullets. This method will instantiate a Bullet object and add it to the world.

Through this process, students understand how to implement the shooting logic by separating the relevant steps into methods of the Tower class. The teacher ensures that students grasp the importance of method invocation and object creation, reinforcing their comprehension of these concepts in the context of their game.

5. Task 5.13 - Discuss How Instance of Class Tower Should Interact with the Relevant Objects Using Messages When Shooting (15 minutes)

Objective: The teacher explains how the tower interacts with bullets and other objects using messages.

Concepts to Discuss: Message passing, method calls.

Activity: Students discuss the message-passing mechanism for shooting bullets.

The session begins with the teacher explaining the concept of message passing and its significance in object-oriented programming. The teacher emphasizes how objects in the game communicate with each other using methods, which serve as messages.

To illustrate this, the teacher uses a UML sequence diagram to describe the interactions among the Tower, Bullet, and Arena objects. The diagram visually represents the flow of messages and method calls, helping students understand the sequence of interactions.

Students discuss the detailed process of how the Tower class should interact with bullets and other objects when shooting. The teacher explains that when the Tower decides to shoot, it sends a message (method call) to create a Bullet instance and add it in Arena. This interaction is initiated from within the act() method of the Tower class. The teacher demonstrates using the UML sequence diagram how the Tower sends a message to the Greenfoot framework to add a new Bullet object to the world. The Tower then sends a message to the Bullet instance, setting its direction to match the Tower's current rotation. This ensures that the bullet moves in the intended direction. Once the Bullet is created and positioned, it will interact with other objects in the game, such as enemies or the edges of the world. The teacher explains how these interactions are handled by the Bullet's act() method, which may involve checking for collisions and removing the bullet when necessary.

Throughout the session, the teacher emphasizes the collaborative nature of these interactions, showing how the algorithm is spread among cooperating objects. This helps students appreciate the modular design and clear communication pathways within their game.

6. Task 5.14 - Implement Shooting of Instance of Class Tower (30 minutes)

Objective: The teacher guides students through the implementation of the shooting mechanism for the Tower class.

Concepts to Discuss: Object creation, actor positioning.

Activity: Students write code to enable the tower to shoot bullets.

The session begins with the teacher explaining the overall goal: to implement the shooting mechanism for the Tower class. The teacher then breaks down the task into manageable steps and guides the students through each one.

First, students prepare the necessary attributes and constructors for the Tower class. The teacher explains that the Tower needs an attribute to keep track of when it can shoot. Next, students create two methods: boolean Tower.canShoot() and void Tower.fire(). Initially, these methods can return false and do nothing, respectively, so they can be used in the act() method. The act() method is then updated to use these methods. The teacher explains that the canShoot() method should return true if the shootCounter reaches 0. The fire() method is implemented to create a Bullet instance and position it correctly.

The teacher ensures that students understand each part of the code, emphasizing the creation of the Bullet object, positioning it in the world, and aligning its rotation with the Tower.

Students then test their solution by running the game, placing a Tower instance, and verifying that it shoots bullets at the appropriate intervals. The teacher encourages

students to troubleshoot any issues, ensuring the bullets are created and move as expected.

By the end of the activity, students have implemented a functioning shooting mechanism for the Tower class, reinforcing their understanding of object creation, actor positioning, and method invocation in Greenfoot.

Commit: [62aec085954beacf996865a55bed312a09c675f2](https://github.com/62aec085954beacf996865a55bed312a09c675f2)

Solution:The solution requires changes to the class Tower.

For start, it is necessary to define new attributes, and proper constructor.

```
11 private int shotDelay;  
12 private int nextShotCounter;  
13  
14 public Tower(int shotDelay)  
15 {  
16     this.shotDelay = shotDelay;  
17     this.nextShotCounter = 0;  
18 }  
19
```

Figure 38. Task 5.14 - 1

Next, new methods should be defined and implemented.

```
38 public boolean canShoot()  
39 {  
40     return this.nextShotCounter == 0;  
41 }  
42  
43 public void fire()  
44 {  
45     Bullet bullet = new Bullet(1);  
46     Arena arena = this.getWorldOfType(Arena.class);  
47     arena.addObject(bullet, this.getX(), this.getY());  
48     bullet.setRotation(this.getRotation());  
49 }
```

Figure 39. Task 5.14 - 2

Finally, the method act() should be implemented.

```

24 public void act()
25 {
26
27     if (this.canShoot())
28     {
29         this.fire();
30         this.nextShotCounter = this.shotDelay;
31     }
32     else
33     {
34         --this.nextShotCounter;
35     }
36 }

```

Figure 40. Task 5.14 - 3

7. Task 5.15 - Towers in Arena (20 minutes)

Objective: The teacher helps students integrate the towers and bullets into the game, creating a functional arena.

Concepts to Discuss: Game integration, testing.

Activity: Students place towers in the arena and test their interactions with bullets and enemies.

The teacher begins by explaining the goal: to integrate towers into the arena and ensure they interact correctly with bullets and enemies. The session will involve placing towers in the arena and testing their behavior within the game environment.

Students start by placing instances of the Tower class in various positions within the arena. The teacher explains how to add towers through the Greenfoot interface, ensuring each tower is correctly positioned.

Next, the teacher introduces the concept of overloading constructors. This is particularly useful for initializing Tower objects with different rotations. The teacher then guides the students through updating the Tower class to include an overloaded constructor that accepts an integer parameter for rotation. This allows for greater control over the placement and orientation of towers within the arena.

Commit: [bfb6a271f490c341c760e654b3f86a87111c54cb](https://github.com/yourusername/yourrepository/commit/bfb6a271f490c341c760e654b3f86a87111c54cb)

Solution: The solution requires changes to the classes: Arena and Tower.

For start, it is necessary to add instances of the Tower class in different places within the arena.

```
38 // TOWERS
39 Tower t1 = new Tower(5, 180);
40 this.addObject(t1, 10, 6);
41 // --
42 Tower t2 = new Tower(5, 270);
43 this.addObject(t2, 16, 11);
44
```

Figure 41. Task 5.15 - 1

After that, change the constructor of the class Tower, and add the new one that overloads existing constructor.

```
14 public Tower(int shotDelay)
15 {
16     this(shotDelay, 0);
17 }
18
19 public Tower(int shotDelay, int rotation)
20 {
21     this.shotDelay = shotDelay;
22     this.nextShotCounter = 0;
23     this.setRotation(rotation);
24 }
```

Figure 42. Task 5.15 - 2

7.4: Greenfoot Objects on a Mission: Bullets, Enemies, and Game Dynamics

Title	Greenfoot Objects on a Mission: Bullets, Enemies, and Game Dynamics
Learning objectives	<p>By the end of the scenario, students will develop proficiency in designing and implementing interactive game dynamics using the Bullet and Enemy classes within the Greenfoot environment. They will understand how to facilitate object interactions through message passing, enabling effective communication between game elements. Students will demonstrate the ability to implement precise collision detection and response mechanisms, specifically detailing how instances of the Bullet class interact with instances of the Enemy class. They will gain practical insights into essential Greenfoot methods such as <code>Greenfoot.showText(String, int, int)</code>, <code>Greenfoot.getRandomNumber(int)</code>, and <code>World.act()</code>, using them to enhance game presentation, introduce randomness, and manage game state updates. Furthermore, students will master the implementation of enemy spawning mechanics and end-of-game conditions, ensuring dynamic gameplay experiences. Through a revision of object associations, students will solidify their understanding of how objects collaborate to create engaging game dynamics, preparing them to apply these skills in future game development projects.</p>
Target audience	<p>Secondary school students attending the OOP4Fun course. Basic programming knowledge including iteration and selection concepts. Students should be introduced to Greenfoot.</p>
Scenario duration	<ol style="list-style-type: none"> 1) Task 5.16 - Discuss how instance of class Bullet should interact with the relevant objects using messages(15 minutes) 2) Task 5.17 - Implement instance of class Bullet hitting instance of class Enemy (30 minutes) 3) Explanation of the code for methods <code>Greenfoot.showText(String, int, int)</code>, <code>Greenfoot.getRandomNumber(int)</code> and <code>World.act()</code> (15 minutes) 4) Task 5.18 - Spawn of enemies and end of the game (30 minutes) 5) Revision of Associations (20 minutes)
Materials & resources	<p>The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.</p>

Description	<p>In this 110-minute learning scenario, secondary school students will dive into the intricacies of game dynamics involving bullets and enemies within the Greenfoot environment. The session focuses on developing students' skills in creating interactive and dynamic gameplay through effective object interactions and game mechanics.</p> <p>The session begins with a 15-minute discussion on how instances of the Bullet class should interact with relevant game objects using messages. This discussion sets the foundation for understanding how objects communicate and collaborate to achieve specific game behaviors.</p> <p>Following this, students will dedicate 30 minutes to implementing the functionality where instances of the Bullet class successfully hit instances of the Enemy class. This task challenges students to apply their understanding of object collision detection and event handling to create impactful interactions within the game.</p> <p>The next segment involves a 15-minute explanation of essential Greenfoot methods: <code>Greenfoot.showText(String, int, int)</code>, <code>Greenfoot.getRandomNumber(int)</code>, and <code>World.act()</code>. Students will gain insights into how these methods contribute to displaying text, generating random numbers for game mechanics, and managing the game world's update cycle.</p> <p>Students then spend 30 minutes on tasks related to enemy spawning and end-of-game conditions. This includes designing and implementing mechanisms for spawning enemies at appropriate intervals and determining conditions for ending the game based on player actions or game objectives.</p> <p>A 20-minute revision session follows, focusing on consolidating understanding of object associations and their roles in implementing game dynamics. Students will review and refine their understanding of how objects interact and collaborate within the Greenfoot environment to achieve desired gameplay effects.</p> <p>At the conclusion of the scenario, students will emerge with a deeper understanding of how to create interactive and engaging game dynamics involving bullets, enemies, and strategic game mechanics within Greenfoot. They will be equipped with practical skills in implementing object interactions, managing game states, and enhancing player experiences through structured game design principles.</p>
Assessment	<p>The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p>
Result	<p>In order to disseminate their results to teachers and fellow students the usual</p>

dissemination	setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.
---------------	---

7.2.2. A Teacher's Guide to Lesson Preparation

1) Task 5.16 - Discuss How Instance of Class Bullet Should Interact with the Relevant Objects Using Messages (15 minutes)

Objective: The teacher leads a discussion on how bullets should interact with other objects, particularly enemies, through message passing.

Concepts to Discuss: Message passing, collision detection, and object interaction.

Activity: The lesson begins with a review of object interactions within the Greenfoot environment, focusing on how instances of different classes communicate and affect each other. The teacher engages students in a discussion to reinforce these concepts and their practical applications in game development.

Students brainstorm and discuss the logic for bullet interactions and the messages they need to send. The teacher begins by explaining the importance of message passing in object-oriented programming. The discussion will focus on how instances of the Bullet class interact with other objects, such as enemies and the arena, particularly when a bullet hits an enemy.

Students are encouraged to brainstorm and share their ideas on the logic for bullet interactions. They consider questions like: What should happen when a bullet hits an enemy? How should the bullet communicate this event to other objects? What messages need to be passed to handle the interaction properly?

The teacher introduces the concept of collision detection, explaining how the game needs to detect when a bullet intersects with an enemy. They also discuss the subsequent actions, such as reducing the enemy's health or removing the enemy from the arena.

To visualize these interactions, the teacher uses a UML sequence diagram. The diagram illustrates the messages exchanged between the Bullet, Enemy, and Arena classes during the interaction.

2) Task 5.17 - Implement Instance of Class Bullet Hitting Instance of Class Enemy (30 minutes)

Objective: The teacher guides students through the implementation of the bullet-enemy interaction.

Concepts to Discuss: Collision detection, method invocation, and object state changes.

Activity: Students write code to handle the collision between a bullet and an enemy, including the effects of the collision. The teacher begins by explaining the concepts of

collision detection and method invocation in Greenfoot, emphasizing how these are crucial for handling interactions between game objects. Students will be guided step-by-step to implement the collision logic between bullets and enemies.

Students will first prepare the necessary attributes and methods in the Bullet and Enemy classes. Students will write code in the Bullet class to detect collision with an Enemy instance and call the Enemy.hit(Bullet) method. Students will then implement the Enemy.hit(Bullet) method to handle the effects of the collision, such as reducing the enemy's health or removing it from the game. Students will test their implementation to ensure the bullet-enemy interaction works as intended.

By the end of the session, students will have a functional collision detection mechanism between bullets and enemies, with appropriate method invocations and object state changes. This exercise reinforces their understanding of collision detection, method calls, and the practical application of these concepts in game development.

Commit: [dcfe31bc006b7f3dcd8b8b759cc1be901c32913c](https://github.com/your-repo/commit/dcfe31bc006b7f3dcd8b8b759cc1be901c32913c)

Solution:The solution requires changes to the classes: Arena, Bullet, Enemy and Tower.

For start, it is necessary to add attribute to the Bullet class, update constructor and define the getter method.

```
9 public class Bullet extends Actor
10 {
11     private int moveDelay;
12     private int nextMoveCounter;
13     private int attack;
14
15     public Bullet(int moveDelay, int attack)
16     {
17         this.moveDelay = moveDelay;
18         this.nextMoveCounter = 0;
19         this.attack = attack;
20     }
21
22     public int getAttack()
23     {
24         return this.attack;
25     }
26 }
```

Figure 43. Task 5.17 - 1

Similarly, add attribute to class Enemy.

```

9 public class Enemy extends Actor
10 {
11     private int moveDelay;
12     private int nextMoveCounter;
13     private int attack;
14     private int hp;
15
16     /**
17      * Creates new enemy.
18      *
19      * @param moveDelay number of act() method calls to make a move.
20      */
21     public Enemy(int moveDelay, int attack, int hp)
22     {
23         this.moveDelay = moveDelay;
24         this.nextMoveCounter = 0;
25         this.attack = attack;
26         this.hp = hp;
27     }

```

Figure 44. Task 5.17 - 2

After that, update the act() method to call the hit() method.

```

44     if (enemy != null) {
45         enemy.hit(this);
46         enemyHit = true;
47     }
48

```

Figure 45. Task 5.17 - 3

Next, add implementation of hit() method of class Enemy.

```

65     public void hit(Bullet bullet)
66     {
67         this.hp -= bullet.getAttack();
68
69         if (this.hp <= 0)
70         {
71             Arena arena = this.getWorldOfType(Arena.class);
72             arena.kill(this);
73         }
74     }

```

Figure 46. Task 5.17 - 4

Then, apply necessary changes to class Arena to call constructor correctly, and implement kill() method.


```

45 // ENEMY
46 Enemy e = new Enemy(2, 1, 2);
47 this.addObject(e, 0, 6);
48 }
49
50 public void respawn(Enemy enemy)
51 {
52     enemy.setLocation(0, 6);
53     enemy.setRotation(0);
54 }
55
56 public void kill(Enemy enemy)
57 {
58     this.removeObject(enemy);
59 }
60 }

```

Figure 47. Task 5.17 - 5

Finally, apply necessary changes to class Tower to call constructor correctly.

```

49 public void fire()
50 {
51     Bullet bullet = new Bullet(0, 1);
52     Arena arena = this.getWorldOfType(Arena.class);
53     arena.addObject(bullet, this.getX(), this.getY());
54     bullet.setRotation(this.getRotation());
55 }

```

Figure 48. Task 5.17 - 6

3) Explanation of the Code for Methods:Greenfoot.showText(String, int, int), Greenfoot.getRandomNumber(int), and World.act() (15 minutes)

Objective: The teacher explains the usage of specific Greenfoot methods that will be useful in the game.

Concepts to Discuss: Displaying text, random number generation, and the act method.

Activity: Students learn how to display text on the screen, generate random numbers, and implement game logic in the act method.

The teacher begins by introducing the Greenfoot methods Greenfoot.showText(String, int, int), Greenfoot.getRandomNumber(int), and World.act(). The purpose of these methods is discussed, highlighting their importance in game development for displaying information, creating randomness, and defining behaviors.

Method `Greenfoot.showText(String, int, int)` is used to display text on the screen at specified coordinates. The teacher explains that this method is useful for showing game information such as scores, health, or instructions directly on the game screen.

Method `Greenfoot.getRandomNumber(int)` generates a random number between 0 (inclusive) and the specified value (exclusive). The teacher discusses how this method can be used to introduce randomness into the game, such as spawning enemies at random locations or generating random movement patterns.

Method `World.act()` is called repeatedly by the Greenfoot framework to execute the main game logic. The teacher emphasizes that the `act` method is where the main game actions and logic are placed, allowing for continuous updates and interactions within the game.

4) Task 5.18 - Spawn of Enemies and End of the Game (30 minutes)

Objective: The teacher helps students implement the spawning of enemies and the end-game conditions.

Concepts to Discuss: Object spawning, game loop, and end-game conditions.

Activity: Students write code to periodically spawn enemies and define the conditions that trigger the end of the game. The teacher begins by explaining the importance of spawning enemies at regular intervals and defining the end-game conditions when all enemies are defeated. The concepts of object spawning, game loops, and end-game conditions are discussed, providing students with a clear understanding of what needs to be implemented.

The `Arena.act()` method will be used to call the enemy spawning process periodically. A delay mechanism should be introduced to control the interval between enemy spawns. Create the `spawn()` method in the `Arena` class to handle the actual spawning process. This method will create an instance of the `Enemy` class, assign properties to the enemy (e.g., position, attributes), add the enemy instance to the arena.

Students will define and implement end-game condition. When number of enemies drop to zero, the game is over, and player won. To do so, students should maintain an attribute in the `Arena` class to keep track of the number of created enemies. Increment this attribute each time an enemy is spawned and decrement it when an enemy is killed. Method `Arena.kill(Enemy)` should be adjusted to check if all enemies are defeated. If all enemies are killed, stop the game using `Greenfoot.stop()` and display a victory message. Therefore, calling the `Greenfoot.stop()` should be the last command in the method.

At the end, test the spawning mechanism by observing the periodic creation of enemies in the arena. Ensure that the delay between spawns is functioning correctly. Verify the end-game condition by simulating the defeat of all enemies and checking if the game stops with a victory message displayed.

By the end of this session, students will have implemented a functional enemy spawning system and defined clear end-game conditions, reinforcing their understanding of game loops, object management, and condition-based game outcomes.

Commit: [d48341a095561500af6032d5c8f56e201060f9a4](https://github.com/48341a095561500af6032d5c8f56e201060f9a4)

Solution:The solution requires changes to the class Arena.

Define new attributes in Arena class.

```
9 public class Arena extends World
10 {
11     private int enemiesCount;
12     private int nextEnemyCounter;
13 }
```

Figure 49. Task 5.18 - 1

In constructor, initialize attribute values.

```
49 this.addObject(e, 0, 6);
50
51 this.enemiesCount = 0;
52 this.nextEnemyCounter = 0;
53 }
```

Figure 50. Task 5.18 - 2

Implement the act() method to spawn enemies periodically, and implement the spawn() method.

```
55 public void act()
56 {
57     if (this.nextEnemyCounter == 0)
58     {
59         this.spawn();
60         this.nextEnemyCounter = 1 + Greenfoot.getRandomNumber(15);
61     }
62     else
63     {
64         --this.nextEnemyCounter;
65     }
66 }
67
68
69 public void spawn()
70 {
71     Enemy enemy = new Enemy(2, 1, 2);
72     this.addObject(enemy, 0, 6);
73     ++this.enemiesCount;
74 }
```

Figure 51. Task 5.18 - 3

Finally, update the kill() method.

```

82 public void kill(Enemy enemy)
83 {
84     this.removeObject(enemy);
85     --this.enemiesCount;
86     if (this.enemiesCount == 0)
87     {
88         this.showText("All enemies destroyed", this.getWidth() / 2, this.getHeight() / 2);
89         Greenfoot.stop();
90     }
91 }

```

Figure 52. Task 5.18 - 4

5) Revision of Associations (20 minutes)

Objective: The teacher reviews the concept of associations between classes, emphasizing how objects interact and communicate in Greenfoot.

Concepts to Discuss: Associations, object communication, and message passing.

Activity: Students discuss and revise their understanding of associations, drawing connections between different objects and their interactions.

The teacher starts by revisiting the key concepts related to associations between classes. This includes how objects interact, communicate, and pass messages to each other. To facilitate this, the teacher can draw on examples and tasks covered in previous lessons, helping students consolidate their knowledge and understand how these concepts apply within the Greenfoot environment.

Use UML sequence diagrams to visually represent the interactions between different objects. For instance, show the sequence of messages when Bullet hits an Enemy and how the Arena handles the spawning and removal of enemies.

By the end of this session, students will have a reinforced understanding of associations and object interactions within the Greenfoot environment. They will be able to clearly articulate how different objects in their game communicate and collaborate, applying these concepts to their own game development projects.

8. Inheritance

Six teaching scenarios have been created within the Variable and expressions thematic unit.

8.0. Introduction to Inheritance in the Greenfoot Environment

Title	Introduction to Inheritance in the Greenfoot Environment
Learning objectives	By the end of this session, students will be able to understand the concepts of inheritance. Understanding of the examined concepts will be discussed in the context of game development, encouraging creativity, teamwork, and an enthusiastic approach to coding with the Greenfoot tool.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge and basic object-oriented programming knowledge. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none">1. Basic concepts of inheritance (15 minutes)2. Class hierarchy and inheritance (15 minutes)3. Task 6.1 and 6.2: Identification of common properties (15 minutes) and identification of the ancestor class (15 minutes)4. Introduction to abstract classes (5 minutes)5. Task 6.3: Definition of an abstract class in the game (10 minutes)
Materials & resources	The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.
Description	<p>In this 75-minute learning scenario, secondary school students are introduced to object-oriented programming principles related to inheritance through the lens of game development by applying the Greenfoot tool. The session begins with a 15-minute introduction by the teacher on basic inheritance concepts, establishing context related to previous sessions and future game development.</p> <p>This is followed by a 15-minute scenario, during which students and teachers discuss the class hierarchy within their game. In order to explain inheritance-related concepts, classes <code>Orb</code> and <code>Direction</code> are observed. During the next 15-minute task, the identification of common properties for these classes is explored. It is observed that these classes do not act during their lifetime; they simply react to messages. Consequently, a common method for acting, the <code>act()</code> method, is identified. This method will be defined in both classes with an empty body. Based on the identified common properties, in the next 15 minutes a new class <code>PassiveActor</code> containing <code>act()</code> method was implemented.</p> <p>In the next 5-minute scenario, abstract classes are introduced. Abstract classes serve as blueprints for other classes and cannot be instantiated. However, they are essential in designing class hierarchies. Given that the <code>PassiveActor</code> class is a blueprint for acting, it is defined as an abstract class in the next 10-minute</p>

	<p>scenario. Additionally, PassiveActoris established as the ancestor of the Orb and Direction classes, making Orb and Direction its descendants. Since the <i>act()</i> method is already defined in the PassiveActor class, it is removed from the Orb and Direction classes.</p> <p>As a result, by the end of the session, students are introduced to novel concepts related to inheritance.</p>
Assessment	<p>The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p> <p>Considering the importance of the inheritance concepts, the project structure opens possibilities for further discussion and modification. In this context, more classes and their hierarchy can be considered, and additional classes, methods and attributes can be introduced. On the other hand, teacher may adapt this topic to show benefits of inheritance and with it connected universality only on here proposed hierarchies.</p>
Result dissemination	<p>In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.</p>

8.0.1. A teacher's Guide to Lesson Preparation

1. Basic concepts of inheritance

Objective: Establishing context related to previous sessions, introducing and explaining the concept of inheritance through real-life examples, and discuss its benefits.

Concepts to Discuss: Inheritance, Real-world inheritance examples.

Activity: In the introduction section context related to the previous sessions is established. Teacher introduces the concept of inheritance. Teacher should make this concept more relatable to students by using real-life examples (e.g., if parent-child relation is considered, children inherit characteristics from their parents, like hair type, eye color, etc.). The benefits of inheritance should be discussed. These concepts are considered in the context of the Greenfoot Environment and Java programming language.

2. Class hierarchy and inheritance

Objective: Introducing the class hierarchy in the context of inheritance by explaining ancestor and descendant classes, discussing real-life examples and the inheritance of properties, highlighting the benefits of class hierarchy.

Concepts to Discuss: Inheritance, Class Hierarchy, Real-world class hierarchy examples, Benefits of inheritance and class hierarchy.

Activity:Teacher introduces the class hierarchy in the context of inheritance concept. Teacher introduces ancestor class (also known as: super class, parent class) and descendant classes (also known as: subclasses, child classes):

- Previously examined real-life classes can be discussed in this context,
- In this context, it should be discussed that subclasses can inherit properties (i.e., attributes and methods) from the parent class,
- However, it should be discussed that subclasses can incorporate additional properties not available in the parent class,

Benefits of the class hierarchy in the context of inheritance concept should be discussed. It should be explained that in Java programming language each class can have multiple subclasses, but only one parent class. The role of the Object class in the context of class hierarchy and inheritance can be discussed.

3. Task 6.1 and 6.2 : Identification of common properties and identification of the ancestor class

Objective: Identifying common properties in the game classes, finding the ancestor class and implementing a new class in the class hierarchy.

Concepts to Discuss: Inheritance, Class Hierarchy, Implementation of inheritance and class hierarchy in the game development.

Activity:In the context of game development, the Orb and Direction classes are considered. It should be observed that these classes react to messages. Therefore, a common method for acting, the *act()* method, should be identified. Based on the identified common properties, new class *PassiveActor* containing *act()* method should be implemented:

- These classes (*PassiveActor*, *Orb*, and *Direction*) should be used for representing class hierarchy in the context of inheritance,
- Teacher can visually represent class hierarchy by using the hierarchy diagram.
- Teacher alerts the students what changed in the Greenfoot environment when *Actor* is substituted with *PassiveActor* in the class

Commit: [afe617814c07a5d885ed06479bf71deda8725f19](#)

Solution:The solution requires creating new class *PassiveActor*, and changes to the classes *Direction* and *Orb*.

Create class *PassiveActor*.

```

1 import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
2
3 /**
4  * Write a description of class PassiveActor here.
5  *
6  * @author (your name)
7  * @version (a version number or a date)
8  */
9 public class PassiveActor extends Actor
10 {
11     /**
12     * Act - do whatever the PassiveActor wants to do. This method is called whenever
13     * the 'Act' or 'Run' button gets pressed in the environment.
14     */
15     public void act()
16     {
17         // Add your action code here.
18     }
19 }

```

Figure 53. Task 6.1 and 6.2 - 1

Change the definition of class Direction by extending previously defined class PassiveActor.

```

9 public class Direction extends PassiveActor
10 {
11     public Direction()
12     {
13         this(0);

```

Figure 54. Task 6.1 and 6.2 - 2

Similarly, change the definition of class Orb by extending previously defined class PassiveActor.

```

9 public class Orb extends PassiveActor
10 {
11     private int hp;
12
13     public Orb(int hp)
14     {
15         this.hp = hp;

```

Figure 55. Task 6.1 and 6.2 - 3

4. Introduction to abstract classes

Objective: Introducing the concept of abstract classes, discussing their role as blueprints in designing class hierarchies, and explore real-world examples to illustrate their application and specialization into subclasses.

Concepts to Discuss: Inheritance, Class Hierarchy, Abstract classes, Real-world examples of abstract classes in the context of inheritance and class hierarchy.

Activity: The concept of abstract class is introduced by the teacher. It should be discussed that abstract classes serve as blueprints for other classes and cannot be instantiated. However, they are essential in designing class hierarchies. Real-world examples related to

abstract classes and subclasses can be discussed by the teacher and students (e.g., class Computer with basic properties can be defined as an abstract class, and can be specialized to Console, Desktop, Laptop, and Mobile Phone, each with a specific set of properties, etc.). Another example could be geometric figures. Rectangle or triangle can be inherited from abstract class figure. When calculation girt and area of general figure we do not have exact formula. But we have exact formula for rectangle and triangle. Square can be inherited from rectangle. Students should discuss more examples of geometric figures and bodies.

5. Task 6.3: Definition of an abstract class in the game

Objective: Discussing the role of the PassiveActor class, and implementing the class as an abstract.

Concepts to Discuss: Inheritance, Class Hierarchy, Abstract classes, Implementation of an abstract class in the game development.

Activity: The concept of abstract class is considered in the Greenfoot Environment and Java programming language. In the context of game development, the PassiveActor class is a blueprint for acting. Therefore, it is defined as an abstract class and established as the ancestor of the Orb and Direction classes, making Orb and Direction its descendants. Since the *act()* method is already defined in the PassiveActor class, it should be removed from the Orb and Direction classes.

Commit: [f7a5702cae29bf21c9c88620d01ef64e4127c21c](https://github.com/f7a5702cae29bf21c9c88620d01ef64e4127c21c)

Solution: The solution requires changes to the classes PassiveActor.

Change the declaration of class PassiveActor, and define it as abstract.

```
9 public abstract class PassiveActor extends Actor
10 {
11     /**
12      * Act - do whatever the PassiveActor wants to do
13      * the 'Act' or 'Run' button gets pressed in
14      */
15     public void act()
```

Figure 56. Task 6.3

8.1. Inheritance Concepts in the Greenfoot Environment (Part 1)

Title	Inheritance Concepts in the Greenfoot Environment
Learning objectives	By the end of this session, students will understand additional concepts of inheritance. The examined concepts will be discussed in the context of game development, encouraging creativity, teamwork, and an enthusiastic approach to coding in the Greenfoot environment.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge and basic object-oriented programming knowledge. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none"> 1) Task 6.4.: Identification of common properties related to entity movement (15 minutes) 2) Task 6.5.: Definition of an abstract class related to entity movement (15 minutes) 3) Task 6.6.: Identification of class-specific properties related to entity movement (15 minutes) 4) Introduction to the <i>super</i> keyword in the context of inheritance (20 minutes) 5) Task 6.7. : Refactoring code related to entity movement (30 minutes)
Materials & resources	The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.
Description	<p>During this 95-minute learning session, secondary school students are introduced to advanced inheritance-related concepts by applying the Greenfoot tool. The session begins with a 15-minute scenario investigating common properties related to entity movement, focusing on the Bullet and Enemy classes. These classes act similarly during lifetime, they move the same way and afterwards they react to the surroundings.</p> <p>Based on the identified common properties, in the next 15-minute task new abstract class <code>MovingActor</code> containing <code>act()</code> method was implemented. This class is a common ancestor that will implement method <code>act()</code> to move in same way and make subclasses focus on their specific purpose. Additionally, <code>MovingActor</code> is established as the ancestor of the Bullet and Enemy classes, making Bullet and Enemy its descendants.</p> <p>In the next 15-minute scenario class-specific properties related to entity movement are examined. In this context, the <code>act()</code> method of respective classes is investigated, as well as the attributes <code>moveDelay</code> and <code>nextMoveCounter</code>. It can be observed that code of <code>act()</code> method responsible for movement is the same.</p> <p>This is followed by a 20-minute scenario, during which the <i>super</i> keyword in the</p>

	<p>context of inheritance was introduced.</p> <p>In the last 30-minute scenario code refactoring related to entity movement was performed. As a result, previously identified attributes <i>moveDelay</i> and <i>nextMoveCounter</i> from Bullet and Enemy subclasses are moved to the ancestor class MovingActor. In addition, parametric constructor to initialize these attributes is defined in MovingActor class. This constructor with proper parameters was invoked from the Bullet and Enemy subclasses using the <i>super</i> keyword. Furthermore, the code responsible for movement in <i>act()</i> method of subclasses Bullet and Enemy was moved to <i>act()</i> method of MovingActor class, while the rest of the implementation remains unchanged in the subclasses. Finally, parent version of method <i>act()</i> is invoked as first line of method <i>act()</i> in subclasses Bullet and Enemy.</p> <p>As a result, by the end of the session, students are introduced to novel concepts related to inheritance.</p>
Assessment	<p>The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p> <p>Considering the importance of the inheritance concepts, the project structure opens possibilities for further discussion and modification. In this context, more classes and their hierarchy can be considered, and additional classes, methods and attributes can be introduced. On the other hand, teacher may adapt this topic to show benefits of inheritance and with it connected universality only on here proposed hierarchies.</p>
Result dissemination	<p>In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.</p>

8.1.1. A teacher's Guide to Lesson Preparation

1. Task 6.4.: Identification of common properties related to entity movement

Objective: Examining the Bullet and Enemy classes, highlighting their similar behaviors during their lifetimes, particularly how they move and react to their surroundings.

Concepts to Discuss: Inheritance, Class Hierarchy, Abstract classes.

Activity: The focus is on Bullet and Enemy classes, which act similarly during lifetime. It should be observed that these classes move the same way and afterwards react to the surroundings.

2. Task 6.5.: Definition of an abstract class related to entity movement

Objective: Examining the Bullet and Enemy classes, highlighting their similar behaviors during their lifetimes, particularly how they move and react to their surroundings.

Concepts to Discuss: Inheritance, Class Hierarchy, Abstract classes, Implementation of an abstract class in the game development.

Activity:Based on the identified common properties, new abstract class MovingActor containing *act()* method should be implemented. Additionally, MovingActor is established as the ancestor of the Bullet and Enemy classes, making Bullet and Enemy its descendants. It should be discussed that the subclasses inherit common properties from the parent class.The MovingActor class is a blueprint for class design and should be declared as an abstract.

Commit: [43e53b533563ce0a860b294ad9009f77409c48d4](#)

Solution:The solution requires creating new class MovingActor, and changes to the classes Bullet and Enemy.

Create abstract class MovingActor that extends class Actor.

```
1 import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
2
3 /**
4  * Write a description of class MovingActor here.
5  *
6  * @author (your name)
7  * @version (a version number or a date)
8  */
9 public abstract class MovingActor extends Actor
10 {
11     /**
12     * Act - do whatever the MovingActor wants to do. This method is called when
13     * the 'Act' or 'Run' button gets pressed in the environment.
14     */
15     public void act()
16     {
17         // Add your action code here.
18     }
19 }
```

Figure 57. Task 6.5 - 1

Change the definition of class Bullet by extending previously defined class MovingActor.

```
9 public class Bullet extends MovingActor
10 {
11     private int moveDelay;
```

Figure 58. Task 6.5 - 2

Similarly, change the definition of class Enemy by extending previously defined class MovingActor.

```
9 public class Enemy extends MovingActor
10 {
11     private int moveDelay;
12     private int nextMoveCounter;
```

Figure 59. Task 6.5 - 3

3. Task 6.6.: Identification of class-specific properties related to entity movement

Objective: Examining the Bullet and Enemy classes, highlighting their similar behaviors during their lifetimes, particularly how they move and react to their surroundings.

Concepts to Discuss: Inheritance, Class Hierarchy, Abstract classes.

Activity:Class-specific properties related to entity movement are examined. The *act()* method of respective classes is investigated, as well as the attributes *moveDelay* and *nextMoveCounter*. It can be observed that code of *act()* method responsible for movement is the same.

4. Introduction to the *super* keyword in the context of inheritance

Objective: introducing the *super* keyword in the context of inheritance, demonstrating its use to invoke properties from the parent class, and discussing its benefits.

Concepts to Discuss: Inheritance, Class Hierarchy, The *super* keyword in the context of inheritance. Position of *super* statement.

Activity:The teacher introduces the *super* keyword. The *super* keyword in the context of inheritance was introduced:

- *super* keyword can be used in order to invoke constructor from the parent class,
- *super* keyword can be used in order to invoke method from the parent class,
- *super* keyword can be used in order to invoke attribute from the parent class,
- *super* must be first statement

Benefits of using the *super* keyword in the context of inheritance should be discussed.

5. Task 6.7. : Refactoring code related to entity movement

Objective: Refactoring code related to entity movement from Bullet and Enemy subclasses to the ancestor class *MovingActor*.

Concepts to Discuss: Inheritance, Class Hierarchy, The *super* keyword in the context of inheritance.

Activity:Code refactoring related to entity movement was performed. Previously identified attributes *moveDelay* and *nextMoveCounter* from Bullet and Enemy subclasses are moved to the ancestor class *MovingActor*. Parametric constructor to initialize these attributes is defined in *MovingActor* class. This constructor with proper parameters was invoked from the Bullet and Enemy subclasses using the *super* keyword. The code responsible for movement in *act()* method of subclasses Bullet and Enemy was moved to *act()* method of *MovingActor* class, while the rest of the implementation remains unchanged in the subclasses. Finally, parent version of method *act()* is invoked as first line of method *act()* in subclasses Bullet and Enemy. It should be discussed that subclasses can incorporate additional properties not available in the parent class (i.e., different implementation of *act()* method).

Commit: [ca1f010a63445c1847b74259a1c6cd4817121db3](https://github.com/ca1f010a63445c1847b74259a1c6cd4817121db3)

Solution:The solution requires changes to the classes *MovingActor*, *Bullet* and *Enemy*.

Create attributes in abstract class *MovingActor* that were previously part of *Bullet* and *Enemy* classes, and create proper parametrized constructor.

```

9 public abstract class MovingActor extends Actor
10 {
11     private int moveDelay;
12     private int nextMoveCounter;
13
14     public MovingActor(int moveDelay)
15     {
16         this.moveDelay = moveDelay;
17         this.nextMoveCounter = 0;
18     }
19

```

Figure 60. Task 6.7 - 1

Make changes to classes Bullet and Enemy by removing existed attributes that became part of abstract class MovingActor, and change the constructors to call the super constructor.

```

9 public class Bullet extends MovingActor
10 {
11
12     private int attack;
13
14     public Bullet(int moveDelay, int attack)
15     {
16         super(moveDelay);
17         this.attack = attack;
18     }
19

```

Figure 61. Task 6.7 - 2

```

9 public class Enemy extends MovingActor
10 {
11     private int attack;
12     private int hp;
13
14     /**
15      * Creates new enemy.
16      *
17      * @param moveDelay number of act() method calls to make a move.
18      */
19     public Enemy(int moveDelay, int attack, int hp)
20     {
21         super(moveDelay);
22         this.attack = attack;
23         this.hp = hp;
24     }

```

Figure 62. Task 6.7 - 3

Make changes to classes MovingActor by implementing the act method to ensure behavior previously implemented in Bullet and Enemy classes and their act method concerning moving delay.

```

21     * Act - do whatever the MovingActor wants to do.
22     * the 'Act' or 'Run' button gets pressed in the e
23     */
24     public void act()
25     {
26         if (this.nextMoveCounter == 0)
27         {
28             this.move(1);
29             this.nextMoveCounter = this.moveDelay;
30         }
31         else
32         {
33             --this.nextMoveCounter;
34         }
35     }
36 }

```

Figure 63. Task 6.7 - 4

Finally, change the implementation of act methods in classes Bullet and Enemy, by removing part related to movement, and replacing it with super.act call.

```
public void act()
{
    super.act();

    boolean enemyHit = false;

    Enemy enemy = (Enemy)this.getOneIntersectingObject(Enemy.class);
    if (enemy != null)
    {
```

Figure 64. Task 6.7 - 5

```
36 public void act()
37 {
38     super.act();
39
40     Direction direction = (Direction)this.getOneIntersectin
41     if (direction != null)
42     {
```

Figure 65. Task 6.7 - 6

8.2. Inheritance Concepts in the Greenfoot Environment (Part 2)

Title	Inheritance Concepts in the Greenfoot Environment
Learning objectives	By the end of this session, students will understand additional concepts of inheritance. The examined concepts will be discussed in the context of game development, encouraging creativity, teamwork, and an enthusiastic approach to coding in the Greenfoot environment.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge and basic object-oriented programming knowledge. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none"> 1) Task 6.8. : Creation of custom enemies (30 minutes) 2) Introduction to the Liskov Substitution Principle (20 minutes) 3) Task 6.9.: Spawning of custom enemies (20 minutes)
Materials & resources	<p>The textbook from the OOP4Fun project.</p> <p>Resources from OOP4Fun project.</p> <p>Project source code from Github/Gitlab.</p> <p>Internet resources.</p>
Description	<p>During this 70-minute learning session, secondary school students are introduced to advanced inheritance concepts using the Greenfoot tool. The session begins with a 30-minute scenario focusing on the Enemy class, where students create additional subclasses representing different enemies (e.g., Frog and Spider). In this context, images and parameterless constructors (with appropriate invocation of the parent constructor) are defined for each type of enemy.</p> <p>In the next 20-minute scenario, the Liskov Substitution Principle (LSP) is introduced. This principle, part of the SOLID principles of object-oriented design, states that functions that use pointers or references to parent classes should be able to use objects of subclasses.</p> <p>Following this, a 20-minute task is dedicated to spawning custom enemies. The <i>Arena.spawn()</i> method is examined, and custom enemies are created through various decisions and stored in a variable of type Enemy. It is observed that no other code in the application needs to be changed, demonstrating the application of the LSP.</p> <p>As a result, by the end of the session, students are introduced to advanced inheritance concepts and practical applications.</p>
Assessment	<p>The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p> <p>Considering the importance of the inheritance concepts, the project structure opens possibilities for further discussion and modification. In this context, more</p>

	classes and their hierarchy can be considered, and additional classes, methods and attributes can be introduced. On the other hand, teacher may adapt this topic to show benefits of inheritance and with it connected universality only on here proposed hierarchies.
Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. Moodle will be used). The students can continue the discussion on the topic on the forum provided to them via the Learning management tool.

8.2.1. A teacher's Guide to Lesson Preparation

1. Task 6.8. : Creation of custom enemies

Objective: Defining subclasses of the Enemy class, each with images and parameterless constructors that appropriately invoke the parent constructor.

Concepts to Discuss: Inheritance, Class Hierarchy, The super keyword in the context of inheritance.

Activity:The focus in on the Enemy class and definition of additional subclasses representing different enemies (e.g., Frog and Spider). Images and parameterless constructors (with appropriate invocation of the parent constructor) should be defined for each enemy type.

Commit: [b0ac1fbe793548a32f7700c292aed631918c8388](https://github.com/0ac1fbe793548a32f7700c292aed631918c8388)

Solution:The solution requirescreating two classes:Frog and Spider that extends class Enemy.

Create class Frog that extends class Enemy, and implement constructor without parameters.

```

1 import greenfoot.*; // (World, Actor, Greenfoot)
2
3 /**
4  * Write a description of class Frog here.
5  *
6  * @author (your name)
7  * @version (a version number or a date)
8  */
9 public class Frog extends Enemy
10 {
11     public Frog()
12     {
13         super (3, 5, 5);
14     }
15 }

```

Figure 66. Task 6.8 - 1

Create class Spider that extends class Enemy, and implement constructor without parameters.

```

1 import greenfoot.*; // (World, Actor, Greenfoot)
2
3 /**
4  * Write a description of class Spider here.
5  *
6  * @author (your name)
7  * @version (a version number or a date)
8  */
9 public class Spider extends Enemy
10 {
11     public Spider()
12     {
13         super (2, 1, 2);
14     }
15 }

```

Figure 67. Task 6.8 - 1

2. Introduction to the Liskov Substitution Principle

Objective: introducing the Liskov Substitution Principle, discussing real-world, and exploring the benefits of adhering to this principle in the context of inheritance.

Concepts to Discuss: Inheritance, Class Hierarchy, Reference variables, The Liskov substitution principle.

Activity:The Liskov Substitution Principle is introduced. This principle is part of the SOLID principles of object-oriented design. The principle states that functions that use pointers or references to parent classes should be able to use objects of subclasses. Real-world examples should be discussed (e.g., if Computer class is defined as the parent class, and Console, Desktop, Laptop, and Mobile Phone classes are defined as subclasses, the Liskov Substitution Principle says that functions which are using Computer class will also work with all subclasses, without any change in the code). Benefits of using the Liskov Substitution Principle in the context of inheritance should be discussed.

3. Task 6.9.: Spawning of custom enemies

Objective: Demonstrating the application of the Liskov Substitution Principle by creating custom enemies.

Concepts to Discuss: Inheritance, Class Hierarchy, Reference variables, The Liskov substitution principle.

Activity:The task is dedicated to spawning custom enemies. The *Arena.spawn()* method is examined, and custom enemies are created through various decisions and stored in a variable of type Enemy. It should be observed that no other code in the application needs to be changed, demonstrating the application of the Liskov Substitution Principle.

Solution:The solution requires changes to method spawn in class Arena.

Change the implementation of spawn method to create different kinds of enemies.

```
69     public void spawn()
70     {
71         Enemy enemy;
72         if (Greenfoot.getRandomNumber(100) < 20)
73         {
74             enemy = new Frog();
75         }
76         else
77         {
78             enemy = new Spider();
79         }
80         this.addObject(enemy, 0, 6);
81         ++this.enemiesCount;
82     }
```

Figure 68. Task 6.9

Commit: [8cd4397f585ec957bbc18ca98e01823f434a13a6](https://github.com/8cd4397f585ec957bbc18ca98e01823f434a13a6)

8.3. Inheritance Concepts in the Greenfoot Environment (Part 3)

Title	Inheritance Concepts in the Greenfoot Environment
Learning objectives	By the end of this session, students will understand additional concepts of inheritance. The examined concepts will be discussed in the context of game development, encouraging creativity, teamwork, and an enthusiastic approach to coding in the Greenfoot environment.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge and basic object-oriented programming knowledge. Students should be introduced to Greenfoot.
Scenario duration	<ol style="list-style-type: none"> 1) Task 6.10.: Discuss hierarchy of Arenas (20 minutes) 2) Task 6.11 and 6.12.: Make universal Arena (30 minutes) and create DemoArena (15 minutes) 3) Task 6.13.: Create custom arenas (30 minutes) 4) Inheritance theory revision (20 minutes)
Materials & resources	The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.
Description	<p>During this 115-minute learning session, secondary school students are introduced to advanced inheritance concepts using the Greenfoot tool. The session begins with a 20-minute discussion about the Arena hierarchy. Subclasses of Arena are responsible for custom layouts (e.g., positions of Orb and Direction instances, size of the arena). These tasks are performed in the constructors of the subclasses, which set and store spawning positions, rotations, and dimensions of the arena.</p> <p>In the subsequent 30-minute task, a universal Arena class is introduced. Additional attributes (<i>spawnPositionX</i>, <i>spawnPositionY</i>, and <i>spawnRotation</i>) are defined, initialized in the constructor, and used in the <i>spawn()</i> and <i>respawn(Enemy)</i> methods. Attributes related to the arena's dimensions (<i>width</i> and <i>height</i>) are also defined and initialized in the constructor. As the Arena class serves as a blueprint for defining concrete arenas, it is defined as an abstract class.</p> <p>Based on the identified Arena class, the next 15-minute task introduces the DemoArena subclass. The DemoArena constructor is defined, invoking the parent class constructor, and the code responsible for the layout of directions, orbs, and towers is moved from the Arena constructor to the DemoArena constructor. Finally, a new instance of the DemoArena class is created.</p> <p>In the following 30-minute scenario, other innovative subclasses of class Arena are created. Code can be shared with other students in the group.</p>

	<p>Finally, the last 20-minute scenario covers inheritance-related theory.</p> <p>As a result, by the end of the session, students are introduced to advanced inheritance concepts and practical applications.</p>
Assessment	<p>The gamification represents non-formal assessment but will increase the interest, intrinsic motivation and learning outputs of the whole group.</p> <p>Considering the importance of the inheritance concepts, the project structure opens possibilities for further discussion and modification. In this context, more classes and their hierarchy can be considered, and additional classes, methods and attributes can be introduced. On the other hand, teacher may adapt this topic to show benefits of inheritance and with it connected universality only on here proposed hierarchies.</p>
Result dissemination	<p>In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. moodle will be used). The students can continue the discussion on the topic on the forum that is provided to them via the Learning management tool.</p>

8.3.1. A teacher's Guide to Lesson Preparation

1. Task 6.10.: Discuss hierarchy of Arenas

Objective: Exploring the Arena class hierarchy, highlighting how subclasses are responsible for defining custom layouts, and implementing these layouts within their respective constructors.

Concepts to Discuss: Inheritance, Class Hierarchy, Constructors.

Activity: The Arena class hierarchy is discussed, it should be observed that the subclasses of Arena are responsible for custom layouts (e.g., positions of Orb and Direction instances, size of the arena). These tasks are performed in the constructors of the subclasses, which set and store spawning positions, rotations, and dimensions of the arena.

2. Task 6.11. and 6.12.: Make universal Arena and create DemoArena

Objective: Introducing a universal Arena abstract class, defining and initializing a concrete Arena subclass, exploring the Arena class hierarchy.

Concepts to Discuss: Inheritance, Class Hierarchy, Abstract classes, Constructors.

Activity: Based on the previous discussion, a universal Arena class is introduced. Additional attributes (*spawnPositionX*, *spawnPositionY*, and *spawnRotation*) are defined, initialized in the constructor, and used in the *spawn()* and *respawn(Enemy)* methods. Attributes related to the arena's dimensions (*width* and *height*) are also defined and initialized in the constructor. As the Arena class serves as a blueprint for defining concrete arenas, it is defined as an abstract class.

Solution: The solution requires changes to class Arena.

Class Arena should be declared as abstract. Additional attributes (spawnPositionX, spawnPositionY, and spawnRotation) are defined and initialized in the constructor.

```
9 public abstract class Arena extends World
10 {
11     private int enemiesCount;
12     private int nextEnemyCounter;
13     private int spawnPositionX;
14     private int spawnPositionY;
15     private int spawnRotation;
16
17     /**
18      * Constructor for objects of class Arena.
19      *
20      */
21     public Arena(int spawnPositionX, int spawnPositionY, int spawnRotation, int width, int height)
22     {
23         // Create a new arena with 24x12 cells with a cell size of 50x50 pixels.
24         super(width, height, 50);
25
26         this.spawnPositionX = spawnPositionX;
27         this.spawnPositionY = spawnPositionY;
28         this.spawnRotation = spawnRotation;
29
30         // DIRECTIONS
```

Figure 69. Task 6.11 and 6.12 - 1

Change methods `spawn()` and `respawn(Enemy)` to use new defined attributes.

```
76     public void spawn()
77     {
78         Enemy enemy;
79         if (Greenfoot.getRandomNumber(100) < 20)
80         {
81             enemy = new Frog();
82         }
83         else
84         {
85             enemy = new Spider();
86         }
87         this.addObject(enemy, this.spawnPositionX, this.spawnPositionY);
88         enemy.setRotation(this.spawnRotation);
89         ++this.enemiesCount;
90     }
91
92     public void respawn(Enemy enemy)
93     {
94         enemy.setLocation(this.spawnPositionX, this.spawnPositionY);
95         enemy.setRotation(this.spawnRotation);
96     }
```

Figure 70. Task 6.11 and 6.12 - 2

Commit: [e9844d7d9b5f19969618b469ebc907d0fe3c1357](https://github.com/0x08c0/0x08c0/commit/e9844d7d9b5f19969618b469ebc907d0fe3c1357)

Based on the identified Arena class, the DemoArena subclass is defined. The DemoArena constructor is defined, invoking the parent class constructor, and the code responsible for the layout of directions, orbs, and towers is moved from the Arena constructor to the

DemoArena constructor. Finally, a new instance of the DemoArena class is created. To activate Demoarena, right click and select new DemoArena.

Solution:The solution requires creating class DemoArena and changes to class Arena.

Create class DemoArena that extends class Arena, and implement constructor that calls super constructor.

```
9 public class DemoArena extends Arena
10 {
11
12     /**
13     * Constructor for objects of class DemoArena.
14     *
15     */
16     public DemoArena()
17     {
18         super(0, 6, 0, 24, 12);
19
20         // DIRECTIONS
21         Direction d1 = new Direction(270);
22         this.addObject(d1, 8, 6);
23         // --
24         Direction d2 = new Direction();
25         this.addObject(d2, 8, 1);
26         // --
27         Direction d3 = new Direction(90);
28         this.addObject(d3, 16, 1);
29         // --
30         Direction d4 = new Direction();
31         this.addObject(d4, 16, 10);
32
33         // ORB
34         Orb o = new Orb(10);
35         this.addObject(o, 23, 10);
36
37         // TOWERS
38         Tower t1 = new Tower(5, 180);
39         this.addObject(t1, 10, 6);
40         // --
41         Tower t2 = new Tower(5, 270);
42         this.addObject(t2, 16, 11);
43     }
44 }
```

Figure 71. Task 6.11 and 6.12 - 3

Make changes to class Arena by removing parts of constructor that was moved to class DemoArena.


```

17  /**
18   * Constructor for objects of class Arena.
19   *
20   */
21  public Arena(int spawnPositionX, int spawnPositionY, int spawnRotation, int width, int height)
22  {
23      // Create a new arena with 24x12 cells with a cell size of 50x50 pixels.
24      super(width, height, 50);
25
26      this.spawnPositionX = spawnPositionX;
27      this.spawnPositionY = spawnPositionY;
28      this.spawnRotation = spawnRotation;
29
30      this.enemiesCount = 0;
31      this.nextEnemyCounter = 0;
32  }

```

Figure 72. Task 6.11 and 6.12 - 4

Commit: [6a6569774b5735f453a56c7cb2cdbf19d228eae9](https://github.com/6a6569774b5735f453a56c7cb2cdbf19d228eae9)

3. Task 6.13.:Create custom arenas

Objective: Defining and initializing customArena subclasses, exploring the Arena class hierarchy.

Concepts to Discuss: Inheritance, Class Hierarchy, Abstract classes, Constructors.

Activity: Other innovative subclasses of class Arena are created. Code can be shared with other students in the group.

4. Inheritance theory revision

Objective: Reviewing the concept and benefits of inheritance, discussing class hierarchy and abstract classes, exploring the super keyword and the Liskov Substitution Principle along with their advantages, and examining both real-life and game-related inheritance examples and implementations.

Concepts to Discuss: Inheritance, Class Hierarchy, Abstract classes, The super keyword, The Liskov Substitution Principle, Real-life and Game-related inheritance examples and implementations.

Activity: The concept of inheritance is reviewed. Benefits of inheritance are reviewed. The class hierarchy and its benefits in the context of inheritance concept are discussed. The concept of abstract class is reviewed. The *super* keyword in its benefits in the context of inheritance are discussed. The Liskov Substitution Principle is reviewed and benefits of using the Liskov Substitution Principle in the context of inheritance are discussed. Real-life inheritance examples are discussed. Game-related inheritance examples and implementation are discussed.

9. Encapsulation

Two teaching scenarios have been created within the Encapsulation thematic unit.

9.1. Exploring Encapsulation through Game Development with Greenfoot

Title	Exploring Encapsulation through Game Development with Greenfoot
Learning objectives	The purpose of this learning scenario is to introduce encapsulation to the students through further development of TowerDefense game.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including inheritance. Students should be introduced to Greenfoot in general.
Scenario duration	<ol style="list-style-type: none"> 1. Introduction (5 minutes) 2. Task 7.1.: Team Collaboration and Coding (20 minutes) 3. Task 7.2 and 7.3.: Team collaboration (30 minutes) 4. Discussion (35 minutes) 5. Code explanation (25 min) 6. Task 7.4.: Team Formation and Project Assignment (10 minutes)
Materials & resources	<p>The textbook from the OOP4Fun project. Resources from OOP4Fun project. Project source code from Github/Gitlab. Internet resources.</p>
Description	<p>In this session, secondary school students will learn about encapsulation within object-oriented programming (OOP) in the Greenfoot. The session begins with a concise 5-minute introduction where the teacher outlines the objectives.</p> <p>Students start by developing the ManualTower class as a subclass of the Tower class. This activity focuses on defining two constructors that are consistent with the parent class's constructors to ensure proper initialization. They should implement an act() method which first calls the parent class's act() method.</p> <p>Following the class creation, the instructor introduces a boolean attribute isManuallyControlled, initialized to false. Students create a method changeControl(boolean) which toggles the isManuallyControlled state and alters the tower's image accordingly, demonstrating encapsulation by controlling access to the state of an object through methods. Each student should then manually trigger the changeControl() method on instances of ManualTower and observe how the internal state and external representation change.</p> <p>The core of the session is the development of a private method processUserControl() which should detect mouse clicks on the tower instance. When clicked, the method changes the tower's control state and updates its</p>

	orientation based on mouse position, using encapsulation to hide the complex control logic. Students should implement the method and integrate it within the act() method, testing interaction with the game environment to ensure functionality and learning how private methods protect the code from external changes.
Assessment	<p>This activity will enable teachers to give formative assessment feedback based upon the discussions and monitoring of students' flipped classroom and teamwork.</p> <p>The peer-review assessment will be performed online as a part of a homework assignment. This will remind students of important aspects of the exercise, will make them critically assess other students' work, will give them insights into good or not so good solutions of their peers etc, and will increase the overall achievement of learning outcomes. The work in the team-project that the students are working on will also use these learning outcomes and knowledge.</p>
Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. Moodle will be used). The students can continue the discussion on the topic on the forum provided to them via the Learning management tool.

9.1.1. A teacher's Guide to Lesson Preparation

1. Introduction

The teacher should start the previously developed game and observe how different actors behave. Suggest developing another type of tower that can be manually controlled to remove enemies more easily. The user should be able to control one tower at a time. When the tower is clicked, it should become manually controlled. To indicate which tower is manually controlled, the currently controlled tower should have a different appearance.

2. Task 7.1.:Team Collaboration and Coding

Objective:PreparingclassManualTower for the session

Concepts to discussinheritance, classes, constructors

Activity:Since students already know how to make a descendant class, let them form teams and create a ManualTower class as a descendant of the Tower class. Students should implement both the constructors and the act method, ensuring that the super constructors are called from these methods. In this part of the class, students will review the material, apply it, and improve their practical knowledge of inheritance.

Commit: [63a02fa0c5080165cba8b467da08c4b65f31d0a8](#)

Solution: First, add new class ManualTower:

```

public class ManualTower extends Tower
{
    public ManualTower(int shotDelay)
    {
        this(shotDelay, 0);
    }

    public ManualTower(int shotDelay, int rotation)
    {
        super(shotDelay, rotation);
    }

    public void act()
    {
        super.act();
    }
}

```

Figure 73. Task 7.1

3. Task 7.2. and Task 7.3: Team collaboration

Objective:The teacher explains the need for private methods to the students by defining the changeControl() function

Concepts to discuss: methods, classes, attributes, access modifiers

Activity:The teacher should prepare icons for the manually controlled tower. To change the icon of the object programmatically, the teacher should explain to the students how to use the Actor.setImage(String) method. Allow students some time to test this function.

The teacher should discuss with students how to determine whether the tower is manually controlled. Emphasize that it is not only important to change the object's state but also to update its image. Highlight that if a user wants to change the state of a Tower object and only changes the attribute directly, the image will remain the same. This discussion should help students understand the need to change the value of an attribute through a method and to keep attributes private rather than public. Explain to the students that this practice is called encapsulation, where the internal state is hidden, and public methods are used to change that state in a controlled manner.

Allow the students to implement the logic of the function. Let them manually invoke their method and observe changes in the internal state.

Commit: [2257746b7dac5eaab7acc55d6493319230338f3a](https://github.com/2257746b7dac5eaab7acc55d6493319230338f3a)

Solution:First, add isManuallyControlled attribute and set it to false inside of the constructor.

```

public class ManualTower extends Tower
{
    private boolean isManuallyControlled;

    public ManualTower(int shotDelay)
    {
        this(shotDelay, 0);
    }

    public ManualTower(int shotDelay, int rotation)
    {
        super(shotDelay, rotation);
        this.isManuallyControlled = false;
    }

    ...
}

```

Figure 74. Task 7.2 and 7.3

Second, add changeControl(boolean) method which alters the ManualTower's state and changes it's image accordingly.

```

public void changeControl(boolean manual)
{
    this.isManuallyControlled = manual;

    if (this.isManuallyControlled)
    {
        this.setImage("towerManualControlled.png");
    }
    else
    {
        this.setImage("towerManualUncontrolled.png");
    }
}

```

Figure 75. Task 7.2 and 7.3

4. Discussion

Objective: Understanding encapsulation of logic inside a separate method

Concepts to discuss: methods, branching

Activity: The teacher should point out that the tower's state can be changed only by manually invoking the method. Point out that the mouse could be outside the world, in which case the mouse information will be null. Remind students that the act() method is constantly running during the game and that it should check whether the object has been clicked and only then invoke the changeControl() method. Highlight that the logic for processing the control should be encapsulated inside a separate method processUserControl().

5. Code explanation

Objective: Introducing method needed for resolving the problem

Concepts to discuss: methods, Greenfoot environment

Activity: Consider how to change the actor's state by clicking on the object. To implement this, the GreenFoot.mouseClicked(Object) method should be explained. Also, introduce MouseInfo object, which can be used for retrieving informations about the mouse position.

6. Task 7.4.: Team Formation and Project Assignment

Objective: Enhance understanding of private methods and encapsulation through practical assignment

Concepts to discuss: methods, access modifiers, classes

Activity: After defining the processUserControl() private method, let students implement its logic. When the mouse is clicked, the controlled tower should change. If the tower is manually controlled, it should follow and be directed towards the mouse. Remind them that it is possible for mouse to be outside the world. After grouping students into teams, let them implement the logic of the processUserControl() method.

One or two teams will present their work, and the group will discuss the results along with the teacher. By the end of the session, all students should understand how this method is implemented.

Commit: [6ec1f489576019a6493490f9e97797920b923869](#)

Solution: Add private method processUserControl() in the ManualTower class:

```
private void processUserControl()
{
    if (Greenfoot.mouseClicked(this))
    {
        this.changeControl(true);
    }

    if (this.isManuallyControlled)
    {
        MouseInfo mouseInfo = Greenfoot.getMouseInfo();
        if (mouseInfo != null)
        {
            this.turnTowards(mouseInfo.getX(), mouseInfo.getY());
        }
    }
}
```

Figure 76. Task 7.4 - 1

Don't forget to call this function in the act() method of the same class.

```
public void act()
{
    this.processUserControl();
    super.act();
}
```

Figure 77. task 7.4 - 2

9.2. Exploring Encapsulation through Game Development with Greenfoot

Title	Exploring Encapsulation through Game Development with Greenfoot
Learning objectives	The purpose of this learning scenario is to introduce encapsulation to the students through further development of TowerDefense game.
Target audience	Secondary school students attending the OOP4Fun course. Basic programming knowledge including inheritance. Students should be introduced to Greenfoot in general.
Scenario duration	<ol style="list-style-type: none"> 1) Flipped Classroom Session (30 minutes): Students should identify the problem with the previously implemented user control and student should investigate how to solve the problem. 2) Class attributes (5 minutes) 3) Task 7.6.: Add evidence of manually controlled tower (5 minutes) 4) Method of class (10 minutes) 5) Task 7.7.: Change of manually controlled tower from centralized place (20 min) 6) Task 7.8.: Invoke change of manually controlled tower (15 min) 7) Theory revision (10 min)
Materials & resources	<p>The textbook from the OOP4Fun project.</p> <p>Resources from OOP4Fun project.</p> <p>Project source code from Github/Gitlab.</p> <p>Internet resources.</p>
Description	<p>The session starts with students discussing and finding issues related to user control such as the inability to deselect a tower once selected. Students should be involved in discussion and propose solutions to track the currently controlled tower and modify the ManualTower class to include a mechanism to deselect the tower.</p> <p>Lastly, a class method changeControlledInstance() should be implemented which allows changing the control of towers from a centralized method, enhancing understanding of encapsulation by showing how class methods can manage shared state across instances.</p> <p>This comprehensive educational approach teaches the concept of encapsulation, and it demonstrates its importance and utility in real-world applications, developing problem-solving and collaborative skills among the students.</p>
Assessment	<p>This activity will enable teachers to give formative assessment feedback based upon the discussions and monitoring of students' flipped classroom and teamwork.</p> <p>The work in the team-project that the students are working on will also use these learning outcomes and knowledge.</p>

Result dissemination	In order to disseminate their results to teachers and fellow students the usual setup of Github/Gitlab and Learning management system (e.g. Moodle will be used). The students can continue the discussion on the topic on the forum provided to them via the Learning management tool.
----------------------	---

9.2.1. A teacher's Guide to Lesson Preparation

1. Flipped Classroom Session

Objective: Students should recognize the need to initialize an attribute in a single place

Concepts to discuss: class attributes

Activity: At the beginning of the class, let students identify the problem with the user control. Currently it's not possible to deselect the tower. Encourage students to think about how this problem could be resolved. Clarify that in the game only one tower should be selected at the time. This discussion should lead students to the idea of having one place in program which is initialized only once and it can be accessed from other parts of the programs, from other object and actors.

2. Class attributes

Objective: Introduce basics of class attributes

Concepts to discuss: attributes, class attributes, classes

Activity: Explain what class attributes are: variables that belong to the class itself, rather than instances of the class. Relate this concept to the game scenario discussed earlier, where having a centralized attribute to manage the currently selected tower could solve the issue.

3. Task 7.6.: Add evidence of manually controlled tower

Objective: Practical usage of class attributes and null

Concepts to discuss: attributes, class attributes, classes, access modifiers

Activity: To track which tower is currently selected in the game, add private static attribute-controlledInstance to the ManualTower class and initialize it to null. Static attribute is related to the whole class, not to an object of a class. Hence, defining a static variable will allow us to **determine** whether tower has been selected and, if so, which one, by **referencing the class name**, without **needing to access a specific** object. The teacher should emphasize that there is one controlledInstance for the whole game. At the beginning, controlledInstance should be initialized to null, as there is no selected tower. Inspect the internal state of class. Here the teacher explains differences between static and non-static attributes. The teacher with students discuss benefits of using static attributes in games. Teacher should also mention here static methods and discuss with the students where using static methods is beneficial.

Commit: [c4739460bed583d2126de066acc6b1149d022990](https://github.com/4739460bed583d2126de066acc6b1149d022990)

Solution: Add private static attribute controlledInstance to the ManualTower class:

```

public class ManualTower extends Tower
{
    private boolean isManuallyControlled;

    private static ManualTower controlledInstance = null;

    //rest of the code

```

Figure 78, Task 7.6

4. Method of class

Objective: Introduce basics of class methods

Concepts to discuss: class methods, class, objects

Activity: Present the concept of class methods that can operate on class-level data. Discuss the need for methods like `changeControlledInstance` to manage switching the currently controlled tower. Emphasize that these methods can be called without needing an instance of the class. For example, school bell rings for everyone at the same time, it doesn't matter who you are, on the other hand, checking a student's homework requires information about that specific student.

5. Task 7.7.: Change of manually controlled tower from centralized place

Objective: Practical usage of class methods

Concepts to discuss: methods, class methods, class attributes

Activity: Teacher should add method `changeControlledInstance` to change manually controlled tower. Parameter of the method will be the tower user wants to select. First, it should be checked whether the controlled instance is currently selected. If it is, nothing should change, but if the passed instance is different than we should change currently controlled instance (reference to the currently controlled instance should be changed). Test out the function manually and observe that the icons of the towers don't change. Point out that only changing the reference of the controlled instance, wouldn't change the control and that it should be done manually. Add the code which releases the currently controlled instance and, after updating the reference, add code which sets manual control of newly controlled instance. Highlight the need for checking null references which could appear if there is no currently controlled instance and if there is no newly controlled instance (when the parameter is null).

Commit: [9dc6d8dd4dcbbd71edb8009c1a72403dea1a0ee0](https://github.com/9dc6d8dd4dcbbd71edb8009c1a72403dea1a0ee0)

Solution: Add new public static method `changeControlledInstance(ManualTower)` to the `ManualTower` class:

```

public static void changeControlledInstance(ManualTower controlledInstance)
{
    if (ManualTower.controlledInstance != controlledInstance)
    {
        // release manual control of previously stored instance
        if (ManualTower.controlledInstance != null)
        {
            ManualTower.controlledInstance.changeControl(false);
        }
        // update reference
        ManualTower.controlledInstance = controlledInstance;
        // set manual control of newly stored instance
        if (ManualTower.controlledInstance != null)
        {
            ManualTower.controlledInstance.changeControl(true);
        }
    }
}

```

Figure 79. Task 7.7

6. Task 7.8.: Invoke change of manually controlled tower

Objective: Practical usage of class methods

Concepts to discuss: methods, class methods, class attributes

Activity: Manually test out the function whether it works correctly. Afterwards, discuss with the students where should this function be invoked. Method should be invoked inside of Arena's act() function and inside of processUserControll() function. Lastly, make method ManualTower.changeControl(Boolean) private and observe changes of instance of ManualTower.

Commit: [c052bbb6aa4c7e690d4d8cf55d3831028fa2b9e3](https://github.com/052bbb6aa4c7e690d4d8cf55d3831028fa2b9e3)

Solution:First, invoke changeControlledInstance(ManualTower) inside of act() method of the Arena class, this will deselect any currently controlled ManualTower:

```

public abstract class Arena extends World
{
    //attributes and constructors

    public void act()
    {
        if (Greenfoot.mouseClicked(this))
        {
            ManualTower.changeControlledInstance(null);
        }

        if (this.nextEnemyCounter == 0)
        {
            this.spawn();
            this.nextEnemyCounter = 1 + Greenfoot.getRandomNumber(15);
        }
        else
        {
            --this.nextEnemyCounter;
        }
    }

    //rest of the functions
}

```

Figure 80. Task 7.8 - 1

In method processUserControl() inside of ManualTower class, instead of the changeControl(boolean) method, invoke ManualTower.changeControlledInstance(ManualTower).

```

private void processUserControl()
{
    if (Greenfoot.mouseClicked(this))
    {
        this.changeControl(true);
        ManualTower.changeControlledInstance(this);
    }

    if (this.isManuallyControlled)
    {
        MouseInfo mouseInfo = Greenfoot.getMouseInfo();
        if (mouseInfo != null)
        {
            this.turnTowards(mouseInfo.getX(), mouseInfo.getY());
        }
    }
}

```

Figure 81. Task 7.8 - 2

7. Theory revision

Summarize the session, highlighting the importance of class attributes and methods in managing game logic efficiently. Encourage students to explore further by applying these concepts in their own programming projects.