Tower defense (OOP4Fun Book)

Tower defense (OOP4Fun Book)

The aim of the course is to learn the Light OOP concepts using the Greenfoot environment. The game Tower defense is used as a demonstration.

Planned ECTS: 2

Number of learners: 5

Mode of delivery: Face to Face

Status: IN PLANNING

Course public access: Public

Contributors:
Zlatko Stapic, Dušan Savić

Course learning outcome	**Level**	**Weight**
Understanding the basic principles of object-oriented programming	Understanding	25
Understanding the basics of algorithmisation	Understanding	25
Understanding the syntax of the Java programming language	Applying	10
Analysing program execution based on the source code	Analysing	20
The ability of creating own programs with the use of OOP	Creating	20
		Total weight: 100

Topic / Unit name	Workload	Learning type	Mode of delivery			Groups	Collaboration	Feedback	Mandatory activity	Assessment		
										Points	Type	Providers

A. Greenfoot environment

The ability of creating own programs with the use of OOP (**100%)**

TS A.1 Introduction to Greenfoot: Exploring Game Development with Creativity

Topic / Unit name	Workload	Learning type	Mode of delivery			Groups	Collaboration	Feedback	Mandatory activity	Assessment
Introduction Teacher introduces today's session, reflects on the previous session and sets challenging goals.	5 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No

Rush-hour challenge 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.).	10 min	Investigation	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Playing games with teacher 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 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 as well as of basic procedures of handling the project files and assets.	30 min	Practice	Onsite	Synchronous	Teacher present	No	No	No	No	No
Team Formation and Project Assignment The students will be grouped in the 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.	5 min	Acquisition	Onsite	Synchronous	Teacher present	Yes	Yes	No	No	No

Team Collaboration and Coding Team collaboration and coding 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.	30 min	Practice	Onsite	Synchronous	Teacher present	Yes	Yes	No	No	No		
Peer Review and Feedback One or two teams will present their work for peer review and feedback and the group will discuss the results along with the teacher.	10 min	Assessment	Onsite	Synchronous	Teacher present	Yes	Yes	No	No	0	Formative	Peer
Homework 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.	30 min	Production	Hybrid	Asynchronous	Teacher present	No	No	No	No	No		
Competition grading As part of the gamification and motivation via competition, each student should vote for 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.).	30 min	Assessment	Onsite	Synchronous	Teacher present	No	No	No	No	0	Summative	Peer
Total unit workload	2.5h											

1. Class definition

Understanding the basic principles of object-oriented programming (**60%)**, Understanding the syntax of the Java programming language (**20%)**, The ability of creating own programs with the use of OOP (**20%)**

TS 1.1: Exploring Classes and Objects through Game Development with Greenfoot

Object Introduce the students with the object concept by real-life examples.	10 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Identification of objects and their properties Students conduct independent research on what objects are to be presented on the stage of the game they are developing (Flipped Classroom Session).	15 min	Investigation	Onsite	Synchronous	Teacher present	No	Yes	Teacher	No	No
Class, instance, inheritance Teacher guided discussion on recognized objects and their classification in classes.	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Orientation in Greenfoot: World, Actor, MyWorld Creating an instance of the world in Greenfoot	10 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Class constructor The teacher presents source code and introduce the concept constructor.	10 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Task 1.2 Prepare world	15 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Image settings How to choose, create, import, paste an image	10 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Task 1.3 Prepare world graphics	15 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No

Total unit workload	1.66h									
TS 1.2. Creating Classes and Objects through Game Development with Greenfoot										
Basic concepts Subclass, object identity, internal state	25 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Task 1.4 Create class Enemy	10 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Task 1.5 Create instance of class Enemy	30 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Interface of object	5 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Message and method	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Task 1.6 Send messages to instance	30 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Theory revision Theory revision of the object, class, instance, internal state, identity, message, method	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	Yes	No	No	No
Total unit workload	2.16h									

2. Algorithm

Understanding the basics of algorithmisation (**60%)**, Understanding the syntax of the Java programming language (**10%)**, Analysing program execution based on the source code (**20%)**, The ability of creating own programs with the use of OOP (**10%)**

TS 2.1. Introduction to Algorithms and Algorithmic Thinking										
Introduction to basic algorithms as a sequence of steps A sequence of steps	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Task 2.1 Write a simple algorithm	20 min	Investigation	Onsite	Synchronous	Teacher present	No	No	Teacher, Peer	No	No

Algorithm and its properties Algorithm and its properties	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Task 2.2 Write a more general algorithm	25 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Peer	No	No
Algorithmisation Algorithmisation	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Total unit workload	1.5h									
TS 2.2. Greenfoot Adventures: Unraveling Java Method Invocation, Documentation, and Application Control										
Explanation of act () method Code explanation	15 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Explanation of move() method Code explanation	10 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Introducing keyword this Basic concepts: Keyword this	5 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Task 2.3 Call a method	10 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Peer, Automated	No	No
Explanation of Autocompleting Code explanation: Autocompleting	5 min	Investigation	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
The importance of code documentation Basic concepts : Documentation comments	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Task 2.4 Add documentation	5 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Task 2.5 Add more documentation	5 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Task 2.6 Read the documentation	10 min	Investigation	Onsite	Synchronous	Teacher present	No	No	Teacher, Peer	No	No

Task 2.7 Explore application controls	20 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Discussion: Algorithm, properties, algorithmisation, Greenfoot buttons Theory revision: Algorithm, properties, algorithmisation, Greenfoot buttons	5 min	Discussion	Onsite	Synchronous	Teacher present	No	Yes	No	No	No
Total unit workload	1.75h									

3. Branching

Understanding the basic principles of object-oriented programming (**10%)**, Understanding the basics of algorithmisation (**60%)**, Understanding the syntax of the Java programming language (**10%)**, Analysing program execution based on the source code (**10%)**, The ability of creating own programs with the use of OOP (**10%)**

TS 3.1. Exploring Branching through Game Development with Greenfoot – Incomplete code branching

Introduction The teacher discusses with the students the concepts that were covered in the previous lesson. Teacher introduces goals for this teaching session.	5 min	Discussion	Onsite	Synchronous	Teacher present	No	No	No	No	No

Code explanation The teacher downloads the latest version of the project: · From Moodle platform · Fromgit 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.	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Incomplete branching 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".	10 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Observing the players' 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, yandrotationattributes in the Enemyclass change when call different methods. How do these values change as you move (up, down, left, and right) and turn?	10 min	Investigation	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Adding world edge detection **Task 3.2:** Teacher asignt 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.	10 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Total unit workload	0.83h									
TS 3.2. Exploring Branching through Game Development with Greenfoot										
Add classes Direction and Orb **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.	30 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No

Collision detection explanation 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.	30 min	Investigation	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
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.	10 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Task 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?	10 min	Practice	Onsite	Synchronous	Teacher present	No	Yes	Teacher, Automated, Peer	No	No

Task 3.6 The teacher assigns the students the task of describing on paper how a pedestrian crosses the street.	15 min	Practice	Onsite	Synchronous	Teacher present	Yes	Yes	Teacher, Automated, Peer	No	No		
Code explanation: Complete branching Complete branching	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No		
Task 3.7 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.	20 min	Practice	Onsite	Synchronous	Teacher present	No	Yes	Teacher, Automated, Peer	No	0	Summative	Teacher, Peer
Task 3.8 Predict enemy movement on custom setup. The teacher puts objects arbitrarily in the World, and the students explain their movement and behavior (independently or in pairs).	30 min	Investigation	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No		
Revision The teacher summarizes the lesson.	5 min	Discussion	Online	Synchronous	Teacher present	No	No	No	No	No		
Total unit workload	2.75h											

4. Variables and expressions

Understanding the basic principles of object-oriented programming (**40%)**, Understanding the basics of algorithmisation (**30%)**, Understanding the syntax of the Java programming language (**20%)**, The ability of creating own programs with the use of OOP (**10%)**

TS 4.1. Introduction to Variables and Data Types in the Greenfoot Environment

Introduction In the introduction section context related to the previous sessions is established. The teacher introduces the term *variable*.	10 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Variable identification § 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).	5 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	Teacher, Peer	No	No
Data types o Teacher introduces the term *data type*, o Examples from real-word 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.), o Data types are considered in the context of the Greenfoot Environment and Java programing language, o Detailed discussion related to variable types required for the game.	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Declaration of variables o Data types are considered in the context of the Greenfoot Environment and Java programing language, o Teacher should explain the difference between declaration and initialization of variables o Declaration of Game-required variables. o Additional examples can be considered. For example, if *act()* method is considered, variable for displaying text can be declared.	10 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No

Initialization of variables o Based on previously presented data types, their data values and data ranges are introduced, o Data values and data ranges are considered in the context of the Greenfoot Environment and Java programing language, o Teacher should explain the difference between declaration and initialization of variables o Initialization of game-required variables. o Additional examples can be considered. For example, if *act()* method is considered, variable for displaying text can be initialized.	5 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Total unit workload	0.75h									
TS 4.2. Introduction to Operators and Expressions in the Greenfoot Environment										
Operators 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.	15 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Arithmetic operators and expressions 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 programing 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.	10 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Boolean operators 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 programing 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.	15 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Relational operators 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 programing 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.	10 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Boolean expressions 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 programing 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.	10 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Object expression 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.	5 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher, Peer	No	No
Reference variable and its null value 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 programing language.The teacher should explain null reference value.	15 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher, Peer	No	No
Task 4.1 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.	15 min	Practice	Onsite	Synchronous	Teacher present	No	No	No	No	No
Total unit workload	1.58h									

TS 4.3. Introduction to Constructors in the Greenfoot Environment										
Basic concepts of constructors 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.	10 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Code explanation 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.	20 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Task: Rename class MyWord to Arena 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()**.	5 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No

Task: Create layout of Arena 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.	30 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Total unit workload	1.08h									
TS 4.4. Introduction to Attributes in the Greenfoot Environment										
Task: Movement-related problem and solution 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.	30 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Attributes The teacher introduces the concept of attributes within the context of Class and Object concepts in Object-Oriented Programming (OOP).	10 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Parameters of constructors The teacher introduces the concept of parameters of the constructorin context of Class and Object concepts in Object-Oriented Programming (OOP).	10 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Task: Enemy.moveDelay 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.	20 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Task: Movement of enemies respecting delay 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.	30 min	Practice	Onsite	Synchronous	Teacher present	No	No	No	No	No
Total unit workload	1.66h									
TS 4.5. Introduction to Constructor Overloading in the Greenfoot Environment										
Basic concepts of constructor overloading The concepts of constructors overloading are discussed.	5 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Task: Parametric constructor of class Direction 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.	25 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Task: Overload constructors in class Direction 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.	25 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Theory revision A review of the previously discussed concepts was conducted during this session.	20 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated	No	No
Total unit workload	1.25h									

5. Association

Understanding the basic principles of object-oriented programming (**30%)**, Understanding the basics of algorithmisation (**30%)**, Understanding the syntax of the Java programming language (**10%)**, The ability of creating own programs with the use of OOP (**30%)**

TS 5.1. Greenfoot Objects on a Mission: Exploring Methods and Associations

Task 5.1 Discuss what should happen when enemy reaches orb.	10 min	Investigation	Onsite	Synchronous	Teacher present	Yes	Yes	Teacher, Automated, Peer	No	No

Task 5.2 Discuss how instance of class Enemy should interact with the relevant objects using messages when hitting instance of class Orb.	15 min	Investigation	Onsite	Synchronous	Teacher present	Yes	Yes	Teacher, Automated, Peer	No	No
Task 5.3 Attribute Enemy.attack and Orb.hp.	10 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Method 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.	15 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Task 5.4 Getter of attribute Enemy.attack.	5 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Task 5.5 Create and test method Arena.respawn(Enemy).	10 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Task 5.6 Create and test method Orb.hit(Enemy).	10 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Total unit workload	1.25h									
TS 5.2. Greenfoot Objects on a Mission: Exploring Associations and Advanced Method Calls										
Association 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.	10 min	Discussion	Onsite	Synchronous	Teacher present	No	Yes	Teacher	No	No
Task 5.7 Call method Orb.hit(Enemy) from Enemy.	15 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Explanation of the code for methods Greenfoot.stop() and World.getWorldOfType(_cls_)	15 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 5.8 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.	30 min	Practice	Onsite	Synchronous	Teacher present	No	Yes	Teacher, Automated, Peer	No	No
Total unit workload	1.16h									
TS 5.3. Greenfoot Objects on a Mission: Towers, Bullets, and Strategic Interactions										
Task 5.9 Create classes Bullet and Tower.	10 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

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 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 5.11 Implement movement of instance of class Bullet	30 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 5.12 Discuss how the instance of class Tower will shoot instance of class Bullet.	15 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 5.13 Discuss how instance of class Tower should interact with the relevant objects using messages when shooting-	15 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 5.14 Implement shooting of instance of class Tower	30 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 5.15 Towers in Arena (20 minutes)	20 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Total unit workload	2.16h									
TS 5.4. Greenfoot Objects on a Mission: Bullets, Enemies, and Game Dynamics										
Task 5.16 Discuss how instance of class Bullet should interact with the relevant objects using messages.	15 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 5.17 Implement instance of class Bullet hitting instance of class Enemy (30 minutes)	30 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Explanation of the code Explanation of the code for methods Greenfoot.showText(String, int, int), Greenfoot.getRandomNumber(int) and World.act()	15 min	Acquisition	Onsite	Synchronous	Teacher present	No	No	No	No	No
Task 5.18 Spawn of enemies and end of the game (30 minutes)	30 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Revision of Associations	20 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Total unit workload	1.83h									

6. Inheritance

Understanding the basic principles of object-oriented programming (**40%)**, Understanding the basics of algorithmisation (**20%)**, Understanding the syntax of the Java programming language (**10%)**, The ability of creating own programs with the use of OOP (**30%)**

6.1. Introduction to Inheritance in the Greenfoot Environment

Basic concepts of inheritance In the introduction section context related to the previous sessions is established. Teacher introduces the concept ofinheritance. 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 programing language.	15 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Class hierarchy and inheritance 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): o Previously examined real-life classes can be discussed in this context, o In this context, it should be discussed that subclasses can inherit properties (i.e., attributes and methods) from theparent class, o 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 conceptshould be discussed. It should be explained that in Java programing 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.	15 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 6.1 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.	15 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No

Task 6.2 Based on the identified common properties, new class PassiveActor containing *act()* method should be implemented: o These classes (PassiveActor, Orb, and Direction) should be used for representing class hierarchy in the context of inheritance, o Teacher can visually represent class hierarchy by using the hierarchy diagram. o Teacher alerts the students what changed in the Greenfootenviroment when Actor in substituted with PassiveActor in the class	15 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No

Introduction to abstract classes 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 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 6.3 Definition of an abstract class in the game. The concept of abstract class is considered in the Greenfoot Environment and Java programing 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.	10 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Peer	No	No

Total unit workload	1.25h									
6.2. Inheritance Concepts in the Greenfoot Environment (Part I)										
Task 6.4 Identification of common properties related to entity movement. The focus is onBullet 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.	15 min	Investigation	Onsite	Synchronous	Teacher present	No	No	Teacher, Peer	No	No
Task 6.5 Definition of an abstract class related to entity movement. 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.TheMovingActor class is a blueprint for class design and should be declared as an abstract.	15 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 6.6 Identification of class-specific properties related to entity movement. 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.	15 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Introduction to the super keyword in the context of inheritance The teacher introduces the *super* keyword. The *super* keyword in the context of inheritance was introduced: o *super* keyword can be used in order to invoke constructor from the parent class, o *super* keyword can be used in order to invoke method from the parent class, o *super* keyword can be used in order to invoke attribute from the parent class, o *super* must be first statement Benefits of using the *super* keyword in the context of inheritance should be discussed. It should be noted that there is a situation, in which this is true - i. e. in contructor.	20 min	Discussion	Onsite	Synchronous	Teacher present	No	No	No	No	No

Task 6.7 Refactoring code related to entity movement. 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).	30 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Total unit workload	1.58h									
6.3. Inheritance Concepts in the Greenfoot Environment (Part II)										
Task 6.8 Creation of custom enemies. 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.	30 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No

Introduction to the Liskov Substitution Principle 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-word 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 Principlein the context of inheritance should be discussed.	20 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 6.9 Spawning of custom enemies. 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.	20 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No
Total unit workload	1.16h									

6.4. Inheritance Concepts in the Greenfoot Environment (Part III)

Task 6.10 Discuss hierarchy of Arenas. 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.	20 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 6.11 Make universal Arena. 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.	30 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
6.12 Create DemoArena. 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.	15 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Task 6.13 Create custom arenas. Other innovative subclasses of class Arena are created. Code can be shared with other students in the group.	30 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Inheritance theory revision 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 Principlein the context of inheritance are discussed. Real-life inheritance examplesarediscussed.Game-related inheritance examples and implementation are discussed.	20 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Total unit workload	1.91h									

7. Encapsulation

Understanding the basic principles of object-oriented programming (**50%)**, Understanding the basics of algorithmisation (**10%)**, Understanding the syntax of the Java programming language (**20%)**, The ability of creating own programs with the use of OOP (**20%)**

7.1. Exploring Encapsulation through Game Development with Greenfoot (Part I)

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.	5 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Task 7.1 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. (Team Collaboration and Coding)	20 min	Production	Onsite	Asynchronous	Teacher not present	No	Yes	Teacher	No	No

Task 7.2 and 7.3 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.	30 min	Production	Onsite	Synchronous	Teacher present	No	Yes	Teacher, Automated, Peer	No	No

Discussion 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 methodprocessUserControl().	35 min	Discussion	Onsite	Synchronous	Teacher present	No	Yes	No	No	No
Code explanation 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 MouseInfoobject, which can be used for retrieving informations about the mouse position.	25 min	Practice	Onsite	Synchronous	Teacher present	No	Yes	No	No	No

Task 7.4 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. (Team Formation and Project Assignment)	10 min	Assessment	Onsite	Synchronous	Teacher present	No	Yes	No	No	0	Formative	Teacher
Total unit workload	2.08h											
7.2 Exploring Encapsulation through Game Development with Greenfoot (Part II)												
Flipped Classroom Session Students should identify the problem with the previously implemented user control and student should investigate how to solve the problem.	30 min	Acquisition	Onsite	Synchronous	Teacher present	No	Yes	Teacher, Automated, Peer	No	No		
Class attributes 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.	5 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No		

Task 7.6 Add evidence of manually controlled tower. **To track which tower is currently selected in the game, a**dd 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 aspecific** object. The teacher should emphasize that there is one controlledInstancefor 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 stati**c** and non-static attributes. The teacher with studentsdiscussbenefits of using static attributes in games. Teacher should also mention here static methods and discuss with the students where usingstatic methods is beneficial.	5 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No

Method of class 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.	10 min	Practice	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No

Task 7.7 Change of manually controlled tower from centralized place. 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).	20 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher, Automated, Peer	No	No

Task 7.8 Invoke change of manually controlled tower. 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.	15 min	Production	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
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.	10 min	Discussion	Onsite	Synchronous	Teacher present	No	No	Teacher	No	No
Total unit workload	1.58h									
Total course workload	**35.5h**									