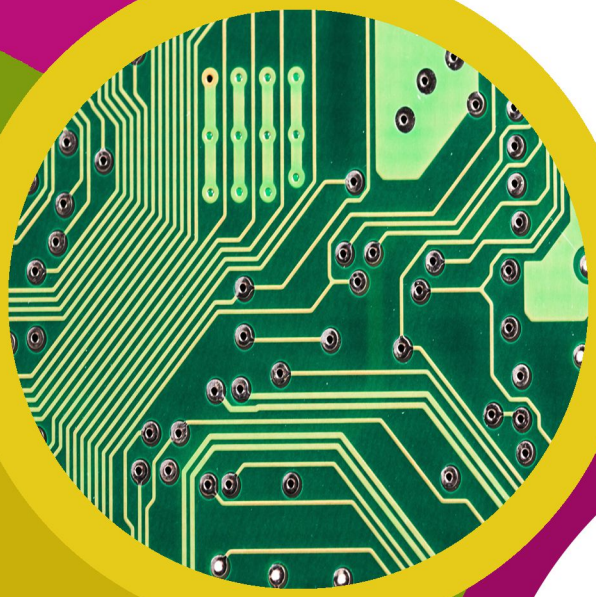
technocamps

Greenfoot Session Plan



UNDEB EWROPEAIDD EUROPEAN UNION
Llywodraeth Cymru Welsh Government
Cronfa Gymdeithasol Ewrop
European Social Fund

Prifysgol Abertawe Swansea University
CARDIFF UNIVERSITY
PRIFYSGOL CAERDYDD
PRIFYSGOL BANGOR UNIVERSITY
Cardiff Metropolitan University
Prifysgol Metropolitan Caerdydd
i.t.wales
PRIFYSGOL ABERYSTWYTH UNIVERSITY
PRIFYSGOL glyndŵr Wrecsam
Wrexham glyndŵr UNIVERSITY
University of South Wales Prifysgol De Cymru

Workshop Schedule

Introduction - 10 minutes

Object-Oriented Programming - 25 minutes

What is Greenfoot? - 5 minutes

Greenfoot - Creating our First Game - 3 hours

Greenfoot - Creating a Jumping Game - 1 hour

Post-Day Questionnaires - 10 minutes

Note: These are estimated times, these will vary between classes, schools etc. so times will need to be adjusted accordingly.

Total: 4 hours 50 minutes

Preparation

1. Ensure all computers have the correct version of Greenfoot (2.4.2) installed and ready to be used.
2. Print out workbooks and cheat sheet.

Learning Outcomes

1. Improved understanding of Object-Oriented programming.
2. Greater experience of programming in Greenfoot.

Attendee Prerequisites

1. No previous Greenfoot experience required.
2. Some knowledge of programming required.

Session Plan Key

In this session plan we use the following colours to differentiate the types of activities:

- **Yellow - Explain.** Teachers should explain the slide/example to the class.
- **Green - Discuss.** Teachers should start an open discussion with the class to get them to feedback some answers/ideas.
- **Purple - Activity.** Students are expected to complete an activity whether it be in their workbooks or on the computer, followed by a discussion of their solutions.
- **Green - Introduction/Conclusion.** The introduction/conclusion is also colour coded green. Teachers should hand out materials in the introduction and conclude the day and collect materials at the end.

Introduction

Begin with introductions, and a brief explanation of the Technocamps programme, before handing out pre-day questionnaires to be filled out by the students and teacher.

Discuss: Computers vs. Humans

Ask the question "Can a computer do anything a human can't?" Give the students 30 seconds to discuss before asking them for an answer. Ask them to explain their answers.

The answer we want to impress on them is that although computers may be much quicker, and they can't make mistakes, there isn't anything that they can do that a human being can't. All we need to do to solve problems like a computer, is to think like a computer. This is called Computational Thinking and it just means breaking down a problem into a set of simple instructions that we can carry out in a certain order.

Activity: Follow These Instructions

The students will all need pen and paper and have to complete a set of instructions by drawing what they are told. They are not allowed to ask questions and not allowed to look at what others have done until the end.

Instructions:

1. "Draw a square"
2. "Inside the square, draw two smaller squares"
3. "On one side of the square, draw a triangle."
4. "Inside the square, on the opposite side of where you drew the triangle, draw a small rectangle."
5. "The final instruction is: Draw a small rectangle which is half inside the triangle, and half outside."

Once they've finished, go around the room and see what each person has ended up with. It is very unlikely that anyone will have drawn a house, but some may have seen the task before.

Activity: Follow These Instructions (continued)

Go through step by step the instructions whilst completing each step on the whiteboard until you've drawn a house. At this point you'll want to ask why didn't anyone draw the house? Whose fault is it? Is it their fault? Is it their teacher's fault? Or is it your fault? And more importantly, ask why?

It is in fact your fault as the instructions you gave weren't simple enough, and they didn't give enough information so the students used their initiative to make a best guess at what to do.

Discuss: Follow These Instructions

Ask who or what else would have the same issues with this kind of task, not being able to ask questions: A computer

Ask what would a computer do?

A computer wouldn't have managed to do the first instruction as it may not know what a square is, and if it did, it wouldn't do anything as we didn't say how big the square needed to be, and where to put it, thickness of the lines etc.

Ask "What if I had just told you all to draw a house instead?"

Not everyone would have drawn the same house, some would have had fantastically detailed houses with thatched roofs, others would have done a very simplistic house.

Explain: Programming Paradigms

Programming Paradigm is a way of classifying programming languages based on their features. In simpler terms, it is defined as the style of programming. There are different programming paradigms.

For example, Python follows a procedural programming paradigm where the programs are composed of functions. Scratch is an event-driven programming language just like any Mobile Application programming.

In an event-driven programming, the program is executed only when an event occurs. Object-oriented programming is the one that Java uses. The concepts of Object-oriented programming will be covered as part of the workshop.

Explain: Procedural Programming

In procedural programming, the program is divided into smaller parts called functions. Functions are more important than data. The functions are written and tested separately and then assembled to form a complete program.

For example, if you hold an account with a bank, the two main functions that you can do with the bank account is to deposit money and to withdraw money. The person who is holding the bank account or the bank where the bank account is held isn't as important to the problem statement as the functions deposit and withdraw money are. While decomposing the problem we identify the various functions that need to be part of the solution and then identify the interactions between the functions and how to assemble them together to automate the solution.

Explain: Object-Oriented Programming

In object-oriented programming, the program is divided into smaller parts called objects. Here, the data is more important than the functions that operate on that data. The solutions are modelled using the real world objects.

For example, in our example of interaction with a bank account, the two main real world objects are the person who has the bank account and the bank account itself. The person will interact with the bank account using the functions deposit or withdraw and they wouldn't need to know how these are implemented. The bank account will in-turn will add or subtract the money based on what function was called by the person. While decomposing a problem we identify the various objects that need to be part of the solution and then identify the interactions between the objects.

Explain: Sequential vs. OOP

Explain the differences between Python which most will be familiar with and object oriented programming. It is a way of programming which is slightly different to how we would usually use Python, it is structured differently to Python's sequential way of coding as Java uses Classes and Objects.

Discuss: Why Java?

Discuss the benefits of Java as described on the slides.

Discuss: Classes and Objects

Ask if anyone knows what Classes and Objects are, and if they have used them before.

Explain: Classes & Objects

A class is a framework or a blueprint for an object. It contains all the information about an object, features, what it can do etc.

An object can only have features and functions which are defined in its class.

For example the class is like a recipe for making ice creams, it gives you all the information you need to make any ice cream, which is the object itself.

A Class describes the contents of the objects that belong to it: it describes the properties and the operations/behaviour. Just like a car. We can describe the properties of a car as transmission type (manual/automatic), steering type (power steering or not), engine placement (front/back), lights, tyres, convertible (yes/no) etc. The behaviour of the car would be drive, park, idle etc.

An Object is an element of a class; objects have the behaviours of their class. The object is the actual component. In our example, a VW Polo, Mini and a VW Beetle are objects of the class car. They all have the behaviour and the properties have values which is unique to each object. For example, a Mini is a convertible where the other two are not. Similarly, the Polo could be an automatic transmission whereas the other two might not be. Each of these cars are an object of the class car.

A class called Student will define all of the things that identifies a student. A student, for example, can be identified by their name, age, and their grade. They are also defined by their behaviour like doHomework and play etc. For example, for the class Student, we can have different objects like Tom, Casey, Luke with different values for their properties but all of them exhibiting the same behaviour.

Activity: Identify Class and Objects

Ask students to identify the class to which the given objects belong to.
Ask students to identify the objects that belong to the class MobilePhones.

Discuss: Inheritance

Ask the students what they think of inheritance in terms of classes? Where have they heard that term and what do they associate the term with in real life?

Explain: Inheritance in Java

Inheritance in Java is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of Object-Oriented programming system. For example, let us assume a class called Box which has two properties named side length and colour. A new class FilledBox which is an inherited class of Box will have the same properties but can also have additional properties like fill and fill colour. If we create objects of the class FilledBox, they will have all the properties belonging to Box and FilledBox. Two examples of such objects with the values filled in are shown.

If we have a class called Student, we can have CS Student which is inherited from Student class and have additional operations/behaviour called convertToBinary which is specific only to the CS Student class. An object of Student class will not have the behaviour convertToBinary. Similarly, if we have another class called French Student which extends from Student class, it can have its own behaviour named translateToFrench.

The subclasses or the inherited classes are more specialised when compared to the parent classes which are general. We use inheritance if we want the classes to be more specific.

Activity: Identify Class and Subclasses

Ask the students to identify the main class or parent class and various subclasses from the given set of objects.

Explain: What is Greenfoot?

Greenfoot is an introductory visual programming environment using the Java programming language.

It is a great language to learn and incorporates a textual experience of programming, providing users with great initial programming knowledge and understanding of basic principles of object-oriented programming.

One of the benefits of using Greenfoot for an introduction into programming is not only the helpful guides and tutorials available, but also the colourful user friendly interface.

The version of Greenfoot used for the workshop is 2.4.2 to ensure consistency with the functions and features used for the workshop.

Explain: World and Actor

Greenfoot has two main classes.
a) World: A world where all actors live and interact. Every instance of the World is different, e.g. Jungle, Space, Sea etc.
b) Actor: An object of the Actor exists in the World. Each actor has its own characteristics and behaviour. E.g. Predator and Prey objects in a Jungle, Aliens and Astronomer objects in Space, and Fish and Shark objects in Sea. Any object of an actor should be added to the World for it to be seen in the world.
A Scenario in Greenfoot is equal to Object(s) of Actor Class + World Class + programming rules.

Activity: Actors and World

Ask students to discuss and write a list of which objects would be of the class Actors and which would be of the World class.

This depends on what these things can do, if they move up and down for example, they cannot be part of the background.

Explain: Greenfoot Coordinate System

The Greenfoot World is based on the given coordinate system. The positive x-axis moves from left to right, just as it does in 8 conventional graphing. The y-axis, however, moves positively from top to bottom unlike the conventional graph.

Explain: Greenfoot Game

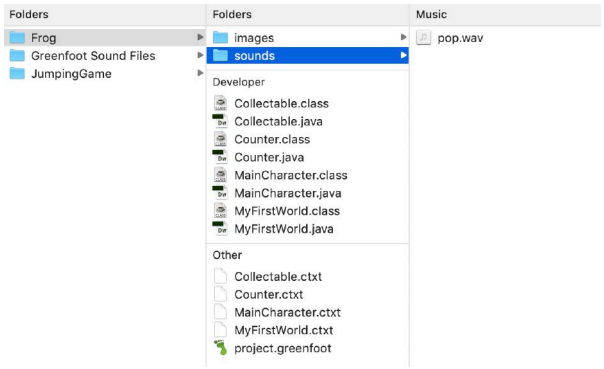
The slides for the workshop lead through this activity step by step. Ensure that you are confident with completing the tasks yourself before delivering the workshop. See the notes on the slides for more explanation.

Explain: Syntax and Naming Conventions

The slides also lists some java syntax that the students need to remember while writing a java program. Some naming conventions are also given to ensure the students write consistent code.

Activity: Adding a Sound

The sound file used in this activity is called pop.wav and can be downloaded from the link on the accompanying slides. This file needs to be placed in the Sound folder of your scenario's folder:



Activity: Random Movement of Collectables

```
setRotation(90*Greenfoot.getRandomNumber(4));
move(1);
```

When added to the act() method, the collectables will constantly pick a random direction and move.

Activity: Adding a Counter

Ask how would you add a different amount to the counter every time a collectable is eaten.

Activity: Quick Quiz

Ask the questions to check how much the students have understood the concept of classes and objects in terms of World and Actors.

Answers: a) 2 , World and Actor b) 1 frog object c) 3 fly objects d) Yes, Score Counter is an object of class Actor e) Four subclasses. One for the World, one for the Frog, one for the Fly and one for the Score Counter.

Explain: Random Number of Objects

A loop in programming is a way to repeat one or more statements. The body of the loop will be repeated while the loop condition is true. Similar to the repeat block in Scratch, Java has while and for loops. We will use for loops to create a random number of objects.

```
for (int i = 0; i < Greenfoot.getRandomNumber(10); i++)
{
   Collectables ant = new Collectables();
  addObject(ant,Greenfoot.getRandomNumber(8),
                        Greenfoot.getRandomNumber(8));
}
```

The for loop has three elements: a) Declaration and initialisation part. We declare a variable and initialise it. b) Condition part. We check if the variable declared has met a condition to continue with the execution of the loop body. c) Change part. We increment the variable every time we complete the execution of the loop body.

The body of the loop should be enclosed within curly braces. In the body of the loop we create a new Collectable object and add it to a random location in the world. The code given should be added in the World subclass.

Activity: Blow Up The Frog

Every time the MainCharacter, or the frog object in our case, eats a Collectable or the fly object, we can increase the size of the frog by a small value.

We can do so, by using the Image object associated with each object. All Actor classes have a common method getImage() which returns an Image object. The Image class in turn has functions like getHeight(), getWidth() and scale(...) which allows you to get and set the width and height of the image associated with the object.

Use the following code to scale the image of the MainCharacter object, a frog in the example, after the removeTouching function is called when the MainCharacter is touching the Collectable class. This will increase the height of the frog image by 10 pixels and the width by 10 pixels giving an impression of blowing up the frog,

```
getImage().scale(getImage().getWidth()+10,
getImage().getHeight()+10);
```

Extension: Creating a Jumping Game

There is a Greenfoot Project in the resources folder called JumpingGame. Load this game and show it to the students. The aim is for them to attempt to create a similar game.

They will need to be shown how to implement jumping and then should be able to finish off the game using what they already know and using the documentation and searching the internet.
Firstly create a subclass of World named Background, then a subclass of Actor named MainCharacter. We will also need to make a subclass of Actor named Platform for the character to run on.

Once we've placed blocks for our floor and placed the character on the blocks, we should be able to right-click the screen and click "Save the world". After doing this, the starting co-ordinate for our character can be found in the Background code. The y-coordinate is going to be very important!

The first thing we'll want to do is set our groundHeight for the MainCharacter. This will be the starting y-coordinate. So we define our private int groundHeight above the act() method and set it to whatever the y coordinate is where we placed it.

Remember, in Greenfoot the y-coordinate increases as we move DOWN the screen.

Next we will need to add movement left and right using the move() method and left and right arrow keys.

If the up key (or space key) is pressed we want our character to jump.
But we also don't want our character to jump if he is already jumping!
So we need to define a couple of boolean variables.

Extension: Creating a Jumping Game (continued)

```
if (getY() == groundHeight)
{
    canJump = true;
} else
{
    canJump = false;
}
```

private boolean canJump = true;
private boolean jumping = false;

Now we need to tell the game when our character is allowed to jump and when it is not. i.e. it can only jump, if it is touching the ground.

This checks the position of our character getY() and compares it to the groundHeight we defined earlier. If it's the same then the character must be on the ground and able to jump. Otherwise, he is in the air and not allowed.
So if the character can jump and the up key is pressed then it will be jumping so we set jumping = true;

```
if (Greenfoot.isKeyDown("up") && canJump)
{
    jumping = true;
}
```

If jumping is true, then we want the character to move upwards to a certain height before stopping.

```
if (jumping) {
    if (getY() == groundHeight - 10) {
        jumping = false;
    } else
    {
        setLocation(getX(), getY() - 2);
    }
}
```

Here they move 2 squares repeatedly until they are 10 squares above the ground before stopping. (**note: the -10 is because the y-coordinates go up as you move DOWN the screen**.)

Extension: Adding Jumping

Once the character has reached the full height of their jump, we need them to fall back down again. So as they are no longer jumping, but they are not on the ground we can use an if statement to capture this:

```
// Checks for jumping and begins falling
if (getY() < groundHeight && !jumping)
{
    setLocation(getX(), getY() + 1);
}
```

Note that our character is falling slower than they jumped up as it is changing by 1 this time. Our character should now be able to jump.

Activity: Finish The Game

From now on the students should be able to add the enemy which moves across the screen (might need help with the logic of how to move the enemy with different speeds every time and how to reset position when it reaches the edge.) Again show them the game whenever they are unsure and ask what they think is happening and how could they implement it?

Discuss: Further Possibilities

If they finish the game, have a discussion with them about how they could develop it, what else could they add or change? Challenge them to attempt those suggestions (adding more enemies, maybe a power-up which will pause the enemies briefly etc.)

technocamps

@Technocamps

Find us on Facebook