Physics Engine

<Author> Mihaela Irina Giurgea
<Author> Corina Lavinia Toma

<Info>

<Keywords> animation, sprite, blocks, loops, graphs, gravitational laws, collisions, free fall, friction force, oblique throwing, motion, momentum, operators, variables

<Disciplines> computer science, physics, mathematics, ICT

<Age level of the students> 14–16

<Hardware> computer

<Language> Scratch[1]

<Programming level> easy, medium

<Summary>

What would you think if we told you that your students could learn two seemingly very different subjects, physics and computer science, more easily and also at the same time? In this unit, the 'engine', i.e. the Scratch[1] programming environment, is the magic tool that will help students to create interesting applications about everyday natural phenomena to better understand the laws of physics and improve their programming skills in the process.

<Conceptual introduction>

Why did we use Scratch[1]? Scratch is a visual programming environment that animates sprites using blocks on the computer screen and helps students to create applications more easily than with classical programming environments (C++, Java, etc.). In addition, our 'engine' helps us to teach two subjects that students consider difficult: physics and computer science. By removing the main obstacles, the impossibility to imagine (see) how a phenomenon actually works and the complicated syntax of coding, we have created an enjoyable new way of teaching.

The students who were involved in the development of this teaching unit already had some coding experience, so they were able to work out how to use Scratch. They use it in their regular as well as in their optional computer science lessons. Besides this, the physics knowledge required is part of the standard curriculum, and it is always useful to revise and apply what you have learned.

<What the students/teachers do>

The unit is all about alternating sequences of learning involving coding and physics.

First, the computer science teacher presented the basics for making a project in Scratch[1]. The students familiarised themselves with several key words related to the Scratch environment: stage, sprites, costumes and movement. You can follow the instructions and explanations for the first application taught in Scratch, an application without physics formulas.[2]

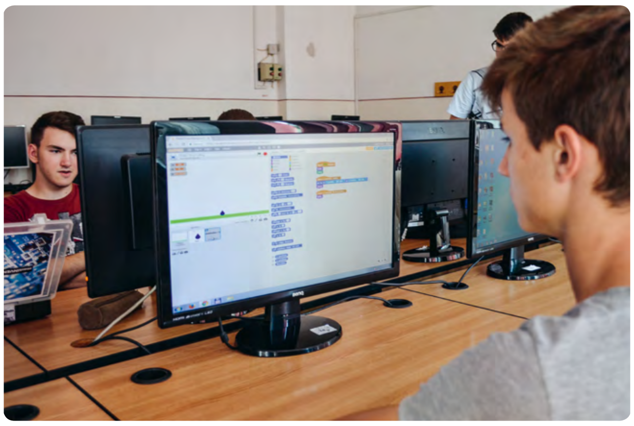
The students needed to understand the interactions between the sprites and their synchronisation as well as how a coordinate system works. You can find a complete tutorial for Scratch online.[3]

To better understand the main algorithms in Scratch, the students looked at exciting and interesting apps. When they saw the code behind these, they were sometimes surprised to discover that they could create such apps themselves.

For the physics part of this unit, the students applied the theoretical principles behind the phenomena of the world around them. For this reason, the physics teacher suggested a wide variety of topics[4], which the students then discussed: formulas needed, possible animation, the design, etc.

The students chose from those topics. A week later, we received apps on the oblique throw of a ball, the free fall of an apple, the more complex fall of a water drop or the collision between two balls, but also the movement of planets in the Solar System or even small games.

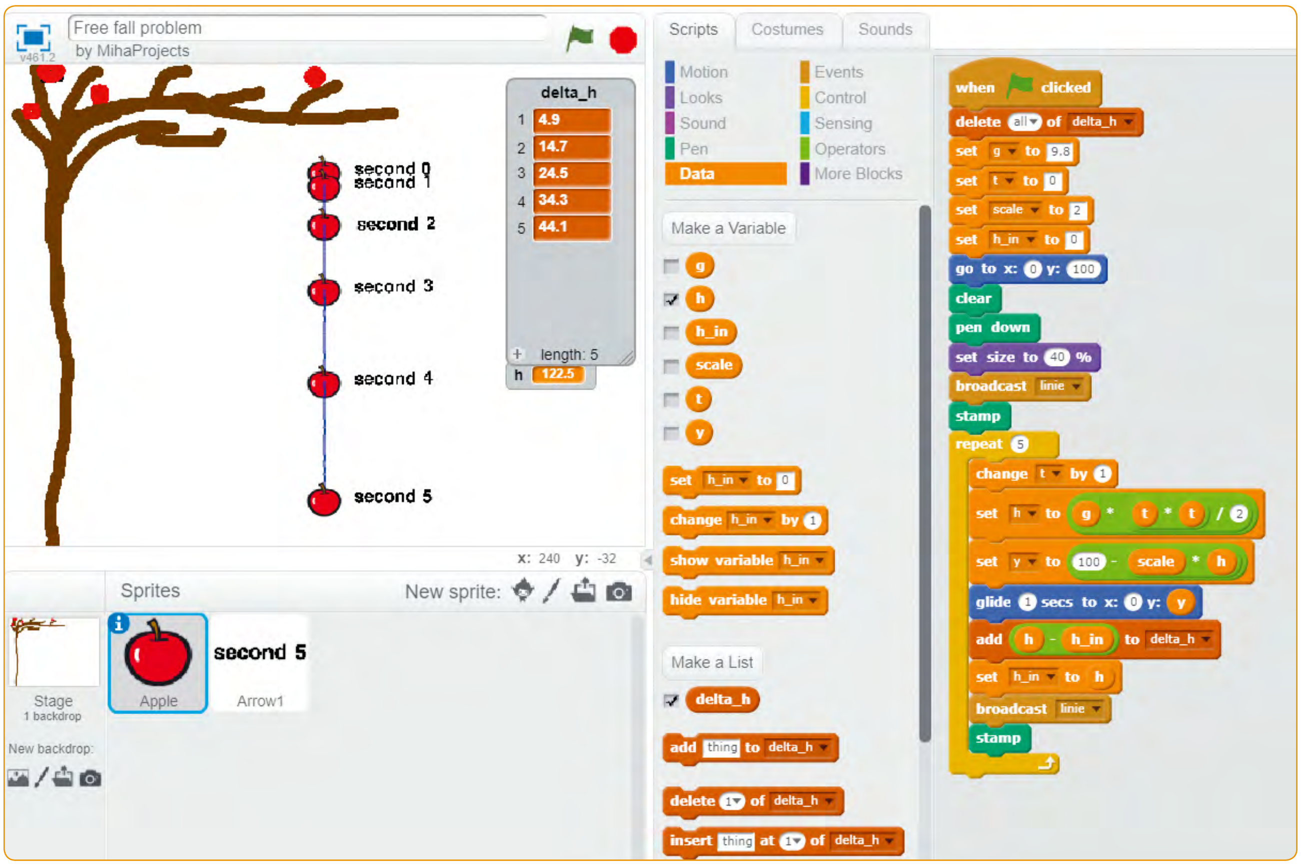
At first, the students worked individually with some help from their teachers. When the projects needed improvement, the students were guided by the teachers and their classmates. (📷1)



📷 1: Individual work

Then, everyone presented their project to the class and received feedback from their peers. This made it easier for the students to work out which parts of the projects needed to be improved: the coding or the physics.

At the end of our project, the older students became teachers for the younger students (12–13 years old) by presenting



```
when [flag] clicked
delete (all) of [delta_h]
set [g] to 9.8
set [t] to 0
set [scale] to 2
set [h_in] to 0
go to x: 0 y: 100
clear
pen down
set size to 40 %
broadcast [linie]
stamp
repeat 5
    change [t] by 1
    set [h] to ((g) * ((t) * (t)) / 2)
    set [y] to (100 - ((scale) * (h)))
    glide 1 secs to x: 0 y: (y)
    add ((h) - (h_in)) to [delta_h]
    set [h_in] to (h)
    broadcast [linie]
    stamp
```

2: Free fall problem

suitable applications and testing them during physics lessons. They enjoyed the responsibility and were very proud of their work. The older students also received suggestions from the younger students. The best of these student simulations are available on the Scratch platform.[2]

The following sections contain examples of how we approached the physics and the programming parts.

<Application 1: Free fall problem (Newton's apple)>

Every student has heard of the free fall of this historical object: Newton's apple.

The application in 2 is inspired by a classical problem: what is the distance travelled every second by Newton's apple in free fall?

Physics theory

We consider a linear motion with constant gravitational acceleration $g = 9.8\,\frac{\text{m}}{\text{s}^2}$.

After a time t, the travelled distance $h(t)$ of the apple is: $h(t) = \frac{gt^2}{2}$.

The initial point $h(0)$ is fixed on the tree branch from which the apple detached itself.

Then we calculate the travelled distance during a longer period, $t+\Delta t$: $h(t+\Delta t) = \frac{g(t+\Delta t)^2}{2}$.

The general formula for the distance $\Delta h(t)$, the distance travelled by the apple during Δt, is:
$\Delta h(t) = h(t+\Delta t) - h(t) = \frac{g(2t\Delta t + \Delta t^2)}{2}$.

Then we use the data of the specific problem to customise the general formula; in this case, 1s for Δt. For the first second, $t = t_{in} = 0$ results in $\Delta h_1 = 4.9$ m, for the next second, t=1s results in $\Delta h_2 = 3 \cdot 4.9$ m $= 14.7$ m and so on. Through mathematical induction, we can calculate the distance travelled during the n^{th} second considering $t = (n-1)$ s:
$\Delta h_n = \frac{9.8(2n-1)}{2}$ m.

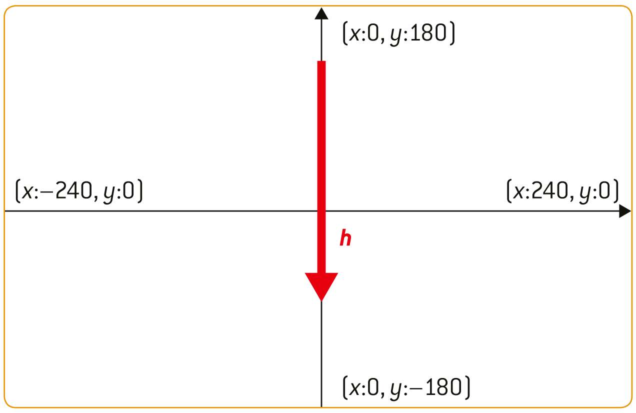
Then the students can calculate, and also see in our animation, that the travelled distance increases by the same amount every second, 9.8 m.

How do we code this?

Variables used:
g: gravitational acceleration
t: a counter for seconds (with values: 0, 1, 2, 3, 4, 5)

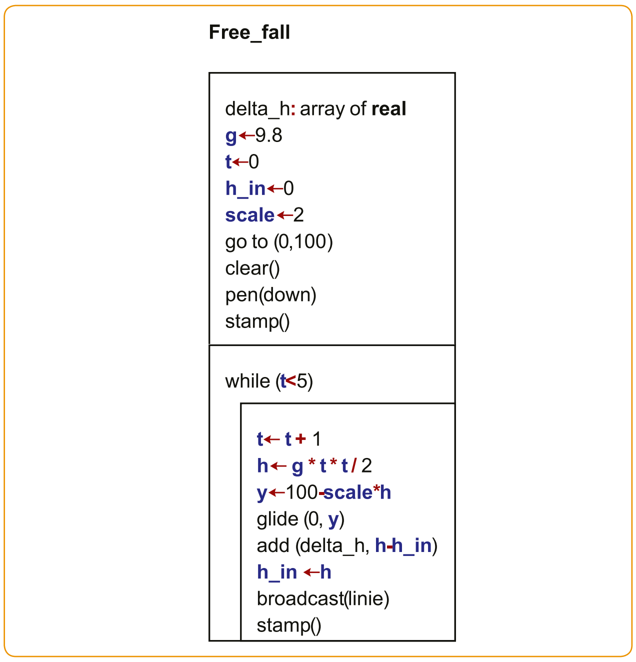
h: the travelled distance after *t* seconds
h_in: the initial position of the apple
delta_h: a list (array) with all distances travelled in every second
y: the *y*-coordinate of the apple
Observation: in this app, the *x*-coordinate remains constant = 0, so you can move the trajectory more easily to the left or the right on the stage.

At the beginning, the apple is at the point with the coordinates (0,180). The initial point and the direction for the travelled distance *h*(*t*) are marked in 📷3.



📷 3: Orientation in the coordinate system

Using a loop 5 times, we recalculated the distance that the apple travelled after every second, working out the new *y*-coordinate and considering the screen's characteristics. See 📷4 for the encoding algorithm.



```
Free_fall
delta_h: array of real
g←9.8
t←0
h_in←0
scale←2
go to (0,100)
clear()
pen(down)
stamp()
while (t<5)
    t← t + 1
    h← g * t * t / 2
    y←100-scale*h
    glide (0, y)
    add (delta_h, h-h_in)
    h_in ←h
    broadcast(linie)
    stamp()
```

📷 4: Free fall problem

Challenge

The students modify Δt and the travelled time *t* (our apple tree should be very tall—it is probably better to draw a tower building) or move the problem to another planet with its own gravitational acceleration. To create a complex project, they could add the friction force from the air and consider a variable gravitational acceleration by dropping the apple from a weather balloon at a higher altitude.

<Application 2: Falling water drop>

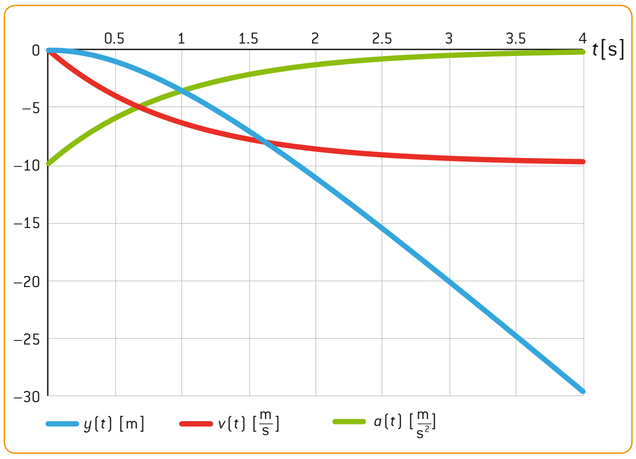
On a rainy day, everyone can observe the fall of water drops. The students analysed the linear movement of one drop with our simulation. At the beginning, they saw that the fall of the water drop is accelerated, but with decreasing acceleration. After some time, the drop's velocity reached its limit, the terminal velocity v_t, when the acceleration reached zero. Then the water drop continued to move at this constant velocity. How can you explain this?

Physics theory

In the accelerated part of the movement, two forces act in opposite directions on the drop: the gravitational force $G = mg$ (m: mass of the drop, g: gravitational acceleration) and the friction force $F_f = kv$ (k: constant of proportionality, v: instantaneous velocity). The acceleration of the drop becomes: $a = g - \frac{k}{m}v$.

In our simulation, we consider a large drop that measures about 5 mm in diameter with the terminal velocity $v_t = 9.8\ \frac{\text{m}}{\text{s}}$.[5] In this case, the constant $\frac{k}{m} = \frac{1}{\text{s}}$. The acceleration decreases when the velocity increases (see 📷5). The initial values are $a = 9.8\ \frac{\text{m}}{\text{s}^2}$, $v = 0$ and $y = 0$.

We use a small program in C++[4] to calculate the instantaneous acceleration and velocity, where we consider the acceleration and the velocity constant for very small time intervals Δt (like 0.05 s). In this case, the velocity increases with $\Delta v = a\Delta t$ and the travelled distance with $\Delta y = v\Delta t$ for each chosen Δt (step-by-step method).

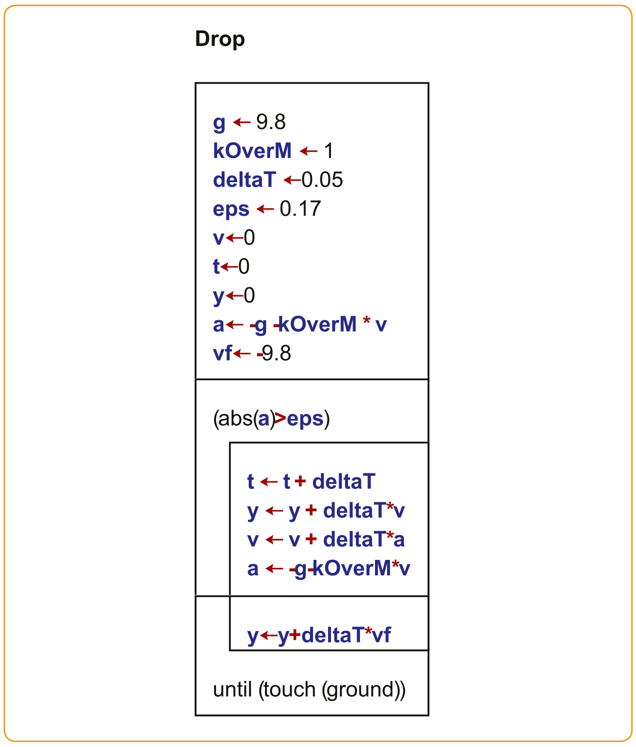


📷 5: Relation between travelled distance, velocity and acceleration

How do we code this?

1. Paint a water drop sprite.
2. Paint a horizontal green line at the bottom of the backdrop.
3. Make a sprite with the message 'acceleration=0', which appears when the acceleration has a value of about 0.
4. Write the code for the drop sprite. The drop starts at the point $(0, y_{\text{init}})$. Using a loop, we recalculate $a(t)$, $v(t)$, $y(t)$. We use the distance reached by the drop after every Δt, working out the new y-coordinate and taking into account the screen's characteristics. The acceleration decreases and when it is about 0, the loop is finished. At this point, the sprite message is shown on the screen. Next, the drop sprite falls at a constant velocity until it touches the green line of the backdrop.

📷6 provides a clear explanation of the code.



```
Drop

g ← 9.8
kOverM ← 1
deltaT ←0.05
eps ← 0.17
v←0
t←0
y←0
a← -g -kOverM * v
vf← -9.8

(abs(a)>eps)
    t ← t + deltaT
    y ← y + deltaT*v
    v ← v + deltaT*a
    a ← -g-kOverM*v

    y←y+deltaT*vf
until (touch (ground))
```

📷 6: Falling drop

Challenge

The students can improve this application if they add a variable for the water drop mass (the water drop diameter can usually take on values from 1 mm to 5 mm)[6] and another type of friction force: $F_f = \frac{kv^2}{2}$.

The students could also add more drops of different masses and compare how they fall.

<Application 3: Elastic collision>

There are many examples of bodies colliding around us. These collisions are complicated, but we consider the elastic collisions with applicability in real life for billiard or steel balls, or in theory for collisions of molecules when the students study the ideal gas model.

Physics theory

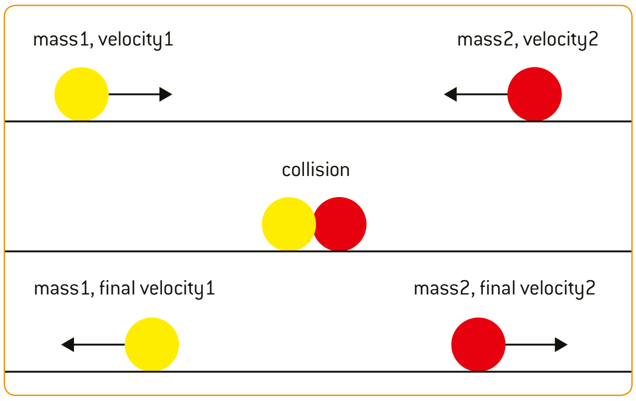
The linear momentum and the kinetic energy are conserved for two balls with the mass m_1 and m_2, the initial velocities $(\vec{v_1})$ and $(\vec{v_2})$ and the final velocities $(\vec{v_{1f}})$and $(\vec{v_{2f}})$. [📷7]

$$m_1\vec{v_1} + m_2\vec{v_2} = m_1\vec{v_{1f}} + m_2\vec{v_{2f}} \quad \text{and}$$

$$\frac{1}{2}m_1v_1^2 + \frac{1}{2}m_2v_2^2 = \frac{1}{2}m_1v_{1f}^2 + \frac{1}{2}m_2v_{2f}^2$$

If all motion takes place along the same line (movement on x-axis), we can use + or – signs to designate directions. Vector notation is not needed for the straight-line collision case, and the final velocities can be calculated with the following equations:

$$v_{1f} = 2\frac{m_1v_1+m_2v_2}{m_1+m_2} - v_1 \quad \text{and} \quad v_{2f} = 2\frac{m_1v_1+m_2v_2}{m_1+m_2} - v_2.$$

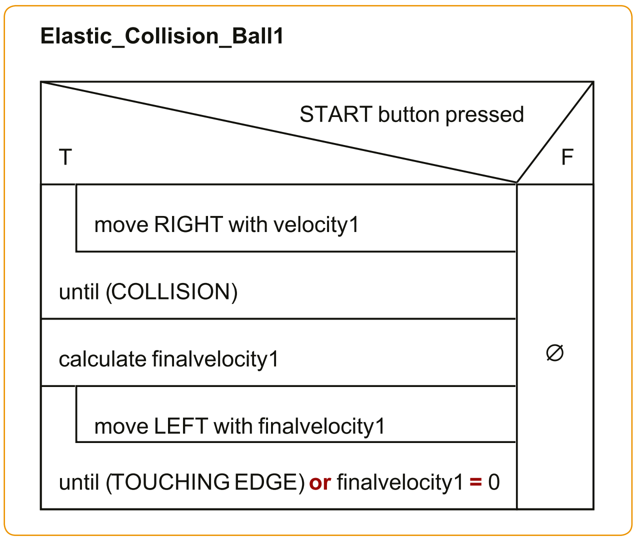


📷 7: Elastic collision

How do we code this?

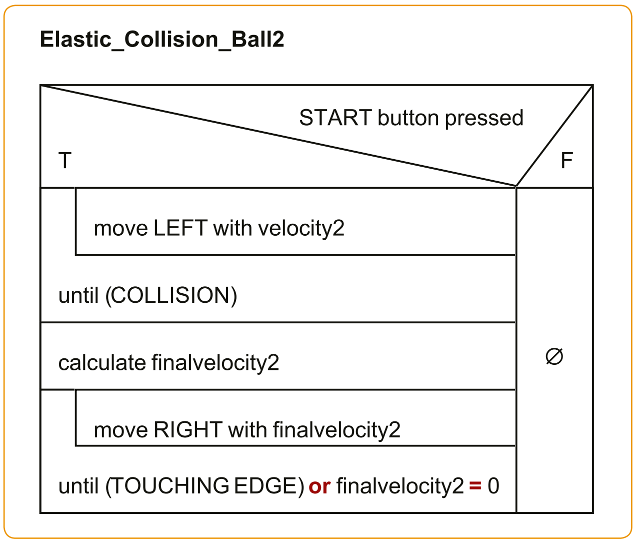
1. Choose two sprites for the balls (Ball1 and Ball2) and one sprite for the **START** button (Start sprite).
2. Use variables: *mass1*, *mass2*, *velocity1*, *velocity2* (the mass and the initial velocity) for each object. Make the variable sliders visible and set the minimum and maximum value for them.
3. Enter the mass and the initial velocities for each object.
4. Press the **START** button. At this time, the sprite broadcasts a message for the ball sprites. When they receive the message, each ball moves towards the other using the well-known formula *distance* = *speed* × *time*.
5. Calculate the final velocities of the balls and use them to move the balls in the right direction until a ball either touches the edge and leaves the scene or stays in its place because its new velocity is 0.

📷8 and 9 provide a clear overview of how the two balls are animated in Scratch[1].



```
Elastic_Collision_Ball1
START button pressed
T:
    move RIGHT with velocity1
  until (COLLISION)
  calculate finalvelocity1
    move LEFT with finalvelocity1
  until (TOUCHING EDGE) or finalvelocity1 = 0
F:
  ∅
```

8: Elastic collision for Ball1



```
Elastic_Collision_Ball2
START button pressed
T:
    move LEFT with velocity2
  until (COLLISION)
  calculate finalvelocity2
    move RIGHT with finalvelocity2
  until (TOUCHING EDGE) or finalvelocity2 = 0
F:
  ∅
```

9: Elastic collision for Ball2

Two examples of using this application:

1. Choose one velocity 0 and equal mass for the balls; after this collision, you will observe that the moving ball stops and the other one moves with the same velocity that the first ball had before the impact.
2. The balls have different velocities and equal mass; after the collision, you will see that the objects take each other's value for the velocity.

In both examples, the balls interchange their momentum.

Challenge

The students could change the size of the balls directly proportional to their mass; they could make an application for a two-dimensional elastic collision (simulation for the Compton effect) or they could program a simulation for the collision of a ball with a wall (reflection law in mechanics).

You could continue the study with another application, i.e. an inelastic collision.[4]

<Conclusion>

<For the students>

Advantages

The students learned physics theory in a more enjoyable way and were able to understand the natural phenomena better using simulations in Scratch. They deepened their computer science and physics knowledge at the same time. Even though their projects were not all perfect, the students clearly improved their coding and algorithmic thinking skills as a result.

Disadvantages

The students worked alone and more at home. They received feedback at school.

<For the teachers>

Advantages

We observed a real interest in creating an original application and learning more than in classical lessons.

Disadvantages

It was difficult for us to coordinate the whole class because of the wide variety of physics topics and the very specific bugs in each application. We think that it would be better to give all the students the same topic and to encourage them to improve it to the best of their varying levels of ability.

<Cooperation activity>

Students from different schools and countries could solve the challenges of the projects and create new ones with other ideas related to the original topic. All these applications could be put in the same place on the Scratch platform, and then a contest could be organised to determine the best of them. Teachers also need to take into account the complexity of the coding and the physics when they evaluate their students' work.

<References>

[1] https://scratch.mit.edu/

[2] All additional materials are available at www.science-on-stage.de/coding-materials.

[3] https://en.scratch-wiki.info/

[4] https://scratch.mit.edu/users/SonS_Coding

[5] http://hypertextbook.com/facts/2007/EvanKaplan.shtml (29/11/2018)

[6] https://journals.ametsoc.org/doi/pdf/10.1175/1520-0450%281969%29008%3C0249%3ATVORA%3E2.0.CO%3B2 (29/11/2018)

<Imprint>

<Taken form>
Coding in STEM Education
www.science-on-stage.eu/coding

<Published by>
Science on Stage Deutschland e.V.
Am Borsigturm 15
13507 Berlin, Germany

<Revision and Translation>
Translation-Probst AG

<Design>
WEBERSUPIRAN.berlin

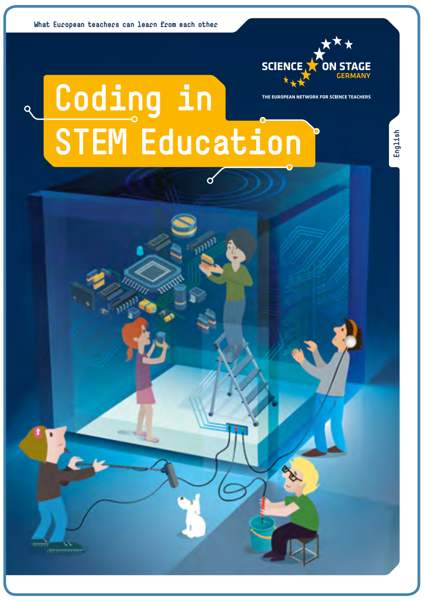
<Illustration>
Rupert Tacke, Tricom Kommunikation und Verlag GmbH

<Credits>
The authors have checked all aspects of copyright for the images and texts used in this publication to the best of their knowledge.

<Please order from>
www.science-on-stage.de
info@science-on-stage.de

<ISBN PDF>
978-3-942524-58-2

First edition published in 2019
© Science on Stage Deutschland e.V.



What European teachers can learn from each other
SCIENCE ON STAGE GERMANY
THE EUROPEAN NETWORK FOR SCIENCE TEACHERS
Coding in STEM Education
English

Science on Stage –
The European Network for Science Teachers

… is a network of and for science, technology, engineering and mathematics (STEM) teachers of all school levels.
… provides a European platform for the exchange of teaching ideas.
… highlights the importance of science and technology in schools and among the public.

The main supporter of Science on Stage is the Federation of German Employers' Association in the Metal and Electrical Engineering Industries (GESAMTMETALL) with its initiative think ING.

Join in – find your country on
www.science-on-stage.eu
www.facebook.com/scienceonstageeurope
www.twitter.com/ScienceOnStage

Subscribe for our newsletter
www.science-on-stage.eu/newsletter

A project by
SCIENCE ON STAGE GERMANY

Main supporter of Science on Stage Germany
think ING.
Die Initiative für Ingenieurnachwuchs

Proudly supported by
SAP®