ICIC Express Letters
Volume **12**, Number **8**, August **2018**

ICIC International ©2018 ISSN 1881-803X
pp. **767–774**

A VISUAL INTERACTIVE EDUCATION SYSTEM ABOUT PHYSICS PROGRAMMING

Juan José Jiménez-Torres, Hiroshi Kamada and Yoshio Yamagishi

Yatsukaho Research Center
Kanazawa Institute of Technology
3-1 Yatsukaho, Hakusan, Ishikawa 924-0838, Japan
jjimenez@ciencias.unam.mx

Received January 2018; accepted April 2018

Abstract. *We present a visual interactive education system about physics programming concept whereby personal computers or smartphones can be used for the student – physics programming educative material interface, allowing for a user self-learning tool. This method is useful for engineering or basic sciences education and teaching using hands-on programming tools. The concept corresponds with a plan to make available self-learning educative materials for physics and engineering. Students benefit by being exposed to self-learning hands-on usage of programming tools. The interactive system concept permits students to code physics programming scripts and work independently. Education institutions also benefit by allowing professors to have an alternative evaluation way outside the traditional classroom. In general, the visual interactive system concept requires few computer skills and less time is spent on coding programs compared to other self-learning programming systems. The concept consists of software which points students how to code a physics simulation and confront their answers with accurate solutions afterwards.*

Keywords: Interactive education systems, Physics programming, Simulations, Answer processing, Answer validation

1. **Background.**

Recent advancements in software have created environments that make visual interactive education systems useful tools for education in basic sciences and engineering. Actually, interactive software is playing a valuable role in the educative process [1]. Currently there are many applications of interactive education systems such as supplements to classroom presentations, laboratories, and dynamic textbooks. Interactive multimedia systems are used, with some success, for educational purposes in developing countries [1]. According to the United Nations Education for All agenda [1] there are clear evidences of the positive impact created from the inclusion of technology into education systems, particularly in developing countries. However, according to UNESCO Education Strategy 2014-2021 [2] there are developing countries reporting inadequate infrastructure and lack of qualified teachers as being barriers to educational development. They argue that the problem of lack of specialists or trained teachers is a significant barrier. [3] shows that this problem exists even in university education. They made a survey of opinion of university professors about virtual learning, and found that personal lack of technology competence was a strong theme. In their survey most professors respondents indicated that their own skill limitations would be a barrier to creating a virtual interactive education course. In that way, [3] suggests that professors must update and develop their technical skills during their academic careers.

In the literature there is an interactive tool created for the conceptual learning of science called Easy Java Simulations: An Interactive Science Learning Tool developed by [4] in 2003. They argue that users concentrate on writing the relations in the model, spend little time on the programming tasks and are able to create their own physical models.

DOI: 10.24507/icicel.12.08.767

Other Java simulations were created by the National Taiwan Normal University Virtual Physics Laboratory. However, [4] explains that these simulations are not completely flexible because models are fixed, which implies that users lose time in the adaptation of the program. Some interactive systems teach programming courses, for example, [5] made a teaching programming system, where users learn how to create algorithms and work on image processing. In this system a basic knowledge of the Java syntax is needed. They argue that programming languages such as C provide the advantage of computational speed, but users waste time with working on basic input or output operations. They also mention that languages such as MATLAB give a good functionality, but many programming skills are needed. Furthermore, commercial systems are expensive and need the accessibility of a supported license [5]. More significantly, in most of these systems users do not have opportunities to code and evaluate their results afterwards.

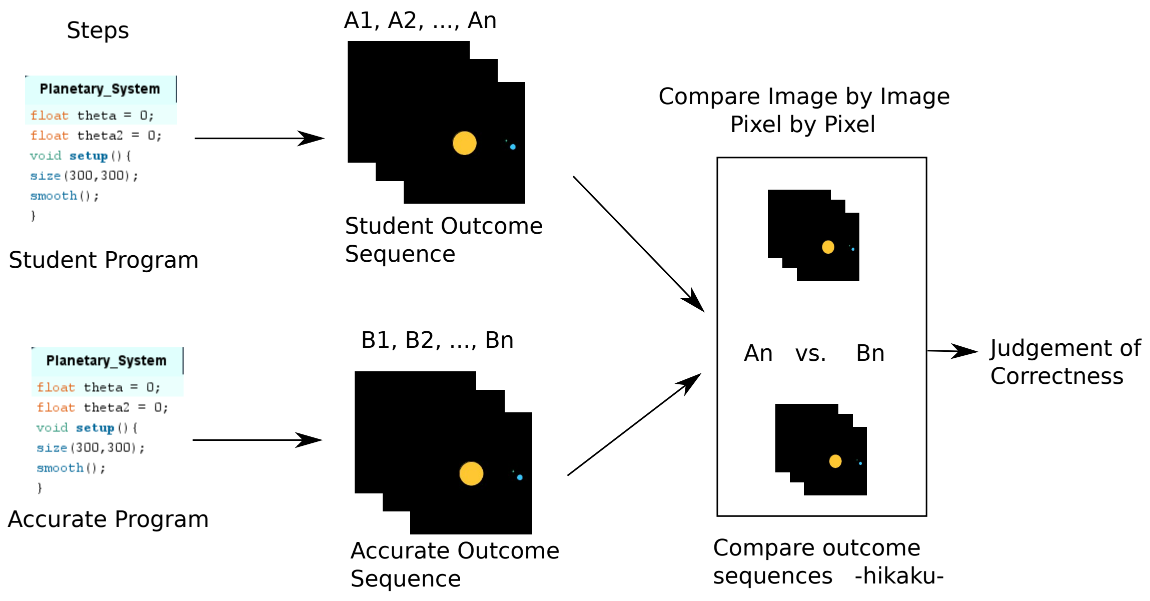
2. **Objectives.** We argue that visual interactive education systems can serve those students who cannot attend classrooms. Many students, particularly in developing countries, find attending hands-on programming courses a not practical option due to lack of resources, demanding professional tasks, and personal responsibilities [6]. In that context we suggest that interactive education systems can represent a good option to cover some students needs. [6] argues that education systems can simulate a computer laboratory where students and professors are located at completely different positions each other. According to [6] the multimedia material gives the chance for connections between teachers and students with better learning materials than material with the printed media. Here we present a visual interactive education system that would help users in their learning processes. The interactive education system concept teaches students how to code a physics simulation and confront their answers with accurate solutions afterwards. Simulations of physical phenomena can help students to understand concepts of basic sciences. They can help students to create explanations for them in terms of models and theories. The visual interactive system concept corresponds to a plan to provide materials and permit students work on self hands-on programming exercises.

3. **Methodology.** We present a visual interactive education system about physics programming concept whereby personal computers can be used for the student – educative material interface, allowing for a user self-learning tool. In this work we have taken account of the basic principles of [7, 8, 9]. The concept corresponds with a plan to make available self-learning hands-on educative materials for physics and engineering. We chose to base our system, the visual user interface and the source code on PROCESSING as the programming language (see [10, 11]). This system lets users focus on coding all steps needed to learn how to create a basic simulation by defining fundamental parameters and creating graphical outputs. Students solve basic exercises making code sequences to modify original code scripts through progressive steps. Then the system provides students opportunities to confront their outputs with accurate simulations.

We present 3 models about classical mechanics, which were selected for their comprehensibility, and generalizability to university students and professors. The intent of this work is to provide a system with a remembering and understanding cognitive level that is useful in the process of teaching, learning and evaluation of physics programming. The contribution and impact of this research directly come from following a technical itinerery in code that users can utilize and adapt in their academic activities and contexts. This interactive system not only uses code as a medium to learn basic physical phenomena in classical mechanics, but also provides contributions in programming skills that will let users understand steps to simulate these phenomena and create useful computational tools.

4. System.

4.1. **System achitecture.** The system consists of two parts. The first part includes an interactive interface, hereafter called *Interface for Instructions*, which shows users instructions to do a proposed exercise. Then users go into a designated programs repository and code scripts located there, which correspond with all steps to learn how to simulate a physics phenomenon. Part 2 consists of an interactive interface, hereafter called *Interface for Comparisons*, which leads users into screens to see if their answers are correct or not. Figure 1 shows how the comparison programs make a judgement of correctness for student's answers. Students code the scripts coresponding with each step for the simulation, and their outcomes are saved as image files into a destined directory. Authors previously calculated all accurate image answers, and these outcome sequences are located in a specific directory. Accurate solutions are based on the PROCESSING literature (see [10, 11]) and Fundamentals of Physics (see [12]). Then *hikaku* programs compare outcome sequences image by image, pixel by pixel, and show a judgement of correctness. In case of presence of mistakes, *hikaku* programs show a message pointing parts where these mistakes could be come from. If the outcome step is correct, the program shows a message indicating that the step is done.

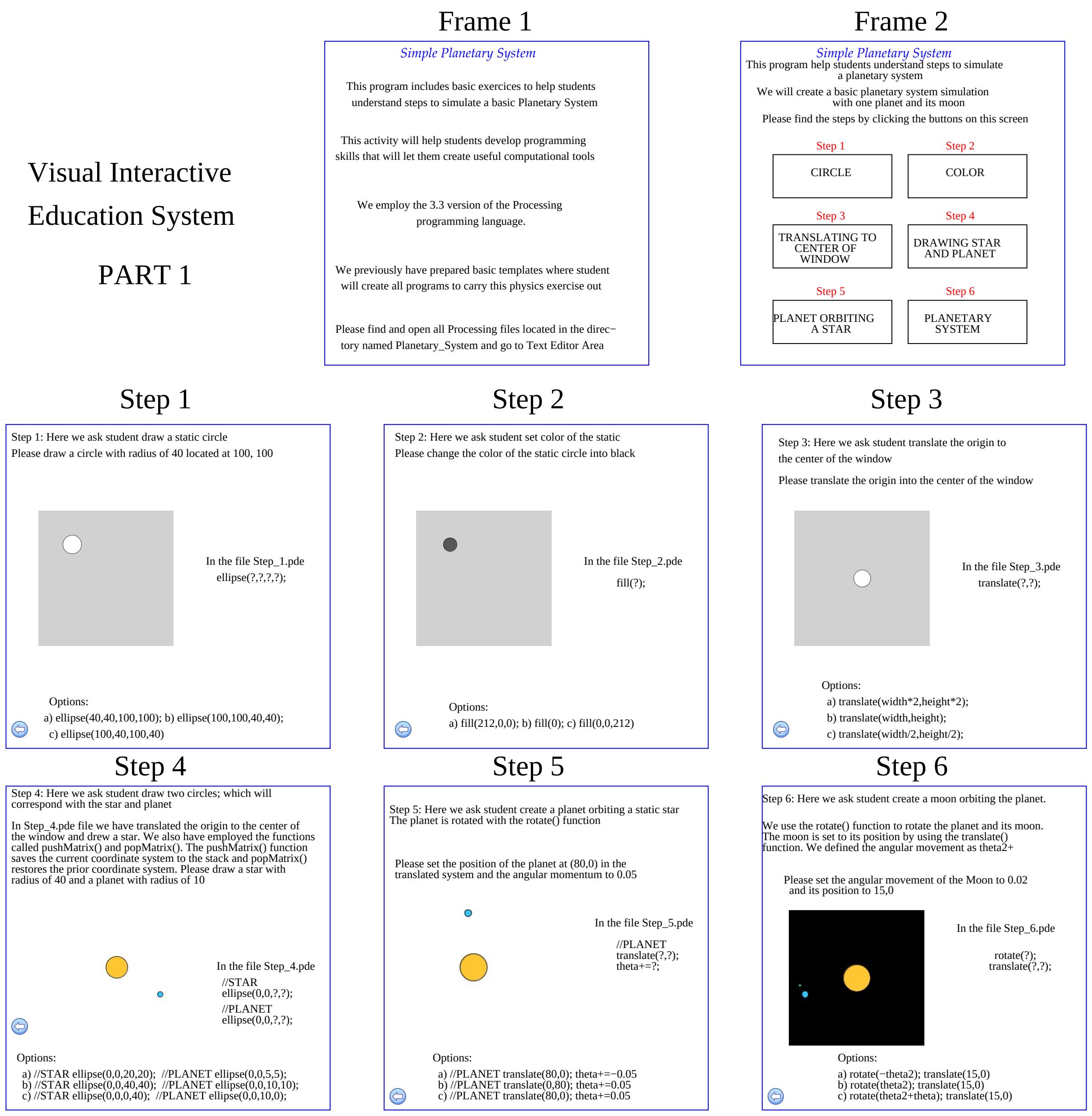


```
Planetary_System
float theta = 0;
float theta2 = 0;
void setup() {
size(300,300);
smooth();
}
```

```
Planetary_System
float theta = 0;
float theta2 = 0;
void setup() {
size(300,300);
smooth();
}
```

FIGURE 1. Structure of the comparison programs called *hikaku*, which confirm if answers from users are correct or not

4.2. **Types of exercises and physics simulations.** The system includes models of physics simulations involving examples of free falling objects, projectile motions and a basic planetary system. These simulations are modeled in a 2D frame following fundamental mechanics laws. These exercises include instructions to help students understand and carry all programming steps out to simulate physical phenomena. This will help students develop programming skills that will let them create useful computational tools. First we take a look into a basic planetary system simulation. By means of six instructions for steps we show user how to simulate a Planet (circle) orbiting a Star (another circle) located at the center of the window. The description of the program and the *Interface for Instructions* for steps are shown in Frames 1 and 2 in Figure 2, respectively, where Part 1 corresponds with instructions for all steps needed to carry the exercise out. They refer to instructions to help students understand methods to simulate a basic planetary system. All instructions for steps and answer options are written in the PROCESSING programming language. This exercise represents a test model, and the circular motion



Visual Interactive Education System

PART 1

Frame 1

Simple Planetary System

This program includes basic exercises to help students understand steps to simulate a basic Planetary System

This activity will help students develop programming skills that will let them create useful computational tools

We employ the 3.3 version of the Processing programming language.

We previously have prepared basic templates where student will create all programs to carry this physics exercise out

Please find and open all Processing files located in the directory named Planetary_System and go to Text Editor Area

Frame 2

Simple Planetary System

This program help students understand steps to simulate a planetary system

We will create a basic planetary system simulation with one planet and its moon

Please find the steps by clicking the buttons on this screen

Step 1

Step 1: Here we ask student draw a static circle
Please draw a circle with radius of 40 located at 100, 100

In the file Step_1.pde
ellipse(?,?,?,?);

Options:
a) ellipse(40,40,100,100); b) ellipse(100,100,40,40);
c) ellipse(100,40,100,40)

Step 2

Step 2: Here we ask student set color of the static
Please change the color of the static circle into black

In the file Step_2.pde
fill(?);

Options:
a) fill(212,0,0); b) fill(0); c) fill(0,0,212)

Step 3

Step 3: Here we ask student translate the origin to the center of the window

Please translate the origin into the center of the window

In the file Step_3.pde
translate(?,?);

Options:
a) translate(width*2,height*2);
b) translate(width,height);
c) translate(width/2,height/2);

Step 4

Step 4: Here we ask student draw two circles; which will correspond with the star and planet

In Step_4.pde file we have translated the origin to the center of the window and drew a star. We also have employed the functions called pushMatrix() and popMatrix(). The pushMatrix() function saves the current coordinate system to the stack and popMatrix() restores the prior coordinate system. Please draw a star with radius of 40 and a planet with radius of 10

In the file Step_4.pde
//STAR
ellipse(0,0,?,?);
//PLANET
ellipse(0,0,?,?);

Options:
a) //STAR ellipse(0,0,20,20); //PLANET ellipse(0,0,5,5);
b) //STAR ellipse(0,0,40,40); //PLANET ellipse(0,0,10,10);
c) //STAR ellipse(0,0,0,40); //PLANET ellipse(0,0,10,0);

Step 5

Step 5: Here we ask student create a planet orbiting a static star
The planet is rotated with the rotate() function

Please set the position of the planet at (80,0) in the translated system and the angular momentum to 0.05

In the file Step_5.pde
//PLANET
translate(?,?);
theta+=?;

Options:
a) //PLANET translate(80,0); theta+=−0.05
b) //PLANET translate(0,80); theta+=0.05
c) //PLANET translate(80,0); theta+=0.05

Step 6

Step 6: Here we ask student create a moon orbiting the planet.

We use the rotate() function to rotate the planet and its moon.
The moon is set to its position by using the translate() function. We defined the angular movement as theta2+

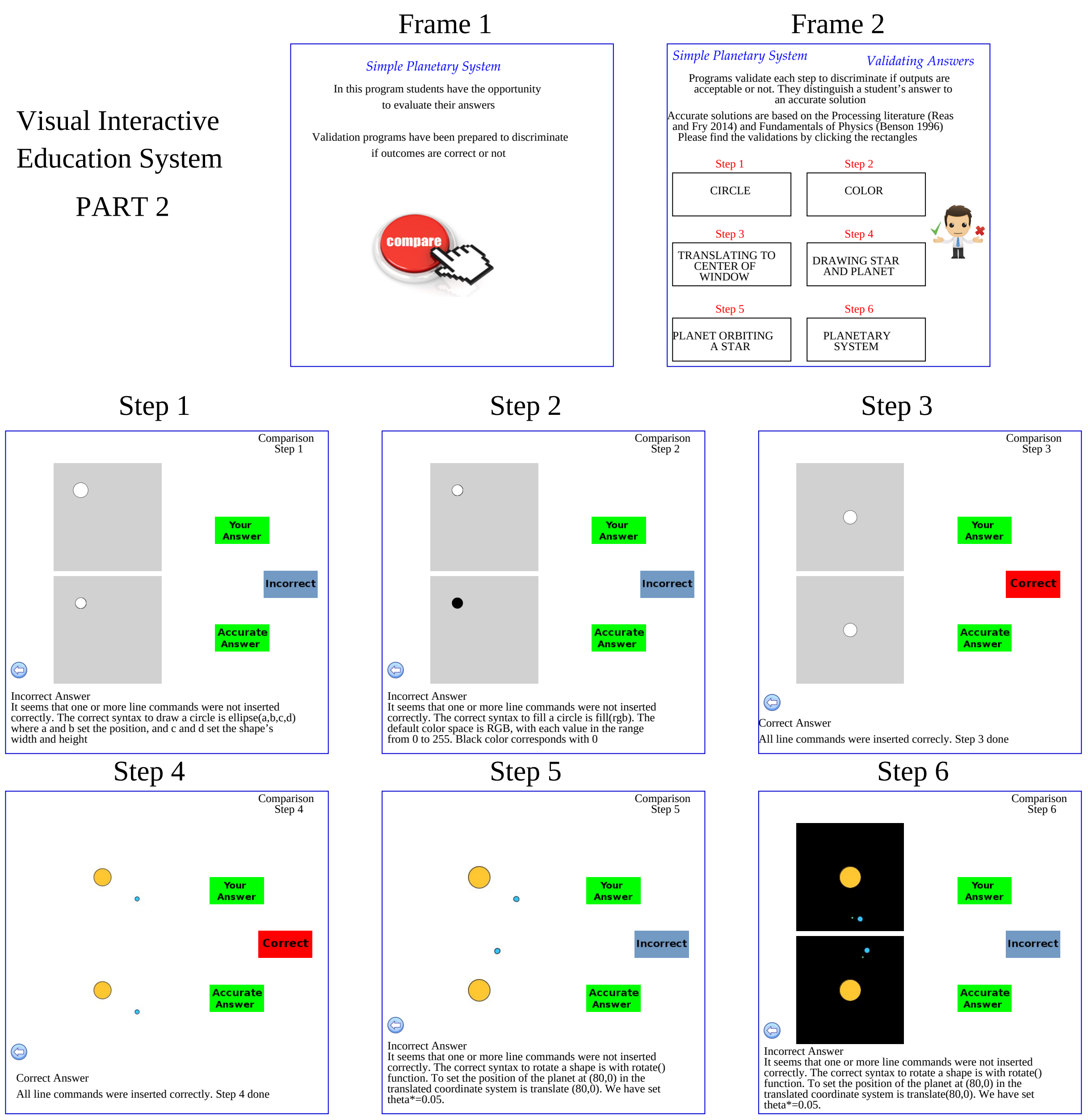
Please set the angular movement of the Moon to 0.02 and its position to 15,0

In the file Step_6.pde
rotate(?);
translate(?,?);

Options:
a) rotate(−theta2); translate(15,0)
b) rotate(theta2); translate(15,0)
c) rotate(theta2+theta); translate(15,0)

FIGURE 2. Part 1 of the interactive system for the basic planetary system simulation. Frame 1 corresponds with the description of the program, frame 2 with the interface for instructions, and other frames with instructions for all six steps. All instructions for steps and answer options are written in the PROCESSING programming language.

is produced by the PROCESSING rotate function. This exercise was tested by a graduate student. Then we confront all student's answers with accurate solutions. Frames 1 and 2 in Figure 3 lead users into screens to see if their answers were correct or not. Part 2 in Figure 3 corresponds with all comparisons between students' outcomes and accurate models for steps 1 to 6. As can be seen in Part 2, in case of presence of mistakes, the programs show messages pointing parts where these mistakes could come from. If the outcome step is correct, the programs show messages indicating that the step is done. On the other hand, the description of the programs and *Interface for Instructions* to get instructions for all steps needed to learn how to simulate projectile motions and free falling objects followed the same structure and philosophy as described in the basic planetary system exercise. These exercises are modeled under the equations of kinematics, where



Simple Planetary System

In this program students have the opportunity to evaluate their answers

Validation programs have been prepared to discriminate if outcomes are correct or not

Simple Planetary System *Validating Answers*

Programs validate each step to discriminate if outputs are acceptable or not. They distinguish a student's answer to an accurate solution

Accurate solutions are based on the Processing literature (Reas and Fry 2014) and Fundamentals of Physics (Benson 1996) Please find the validations by clicking the rectangles

Incorrect Answer
It seems that one or more line commands were not inserted correctly. The correct syntax to draw a circle is ellipse(a,b,c,d) where a and b set the position, and c and d set the shape's width and height

Incorrect Answer
It seems that one or more line commands were not inserted correctly. The correct syntax to fill a circle is fill(rgb). The default color space is RGB, with each value in the range from 0 to 255. Black color corresponds with 0

Correct Answer
All line commands were inserted correcly. Step 3 done

Correct Answer
All line commands were inserted correctly. Step 4 done

Incorrect Answer
It seems that one or more line commands were not inserted correctly. The correct syntax to rotate a shape is with rotate() function. To set the position of the planet at (80,0) in the translated coordinate system is translate (80,0). We have set theta*=0.05.

Incorrect Answer
It seems that one or more line commands were not inserted correctly. The correct syntax to rotate a shape is with rotate() function. To set the position of the planet at (80,0) in the translated coordinate system is translate(80,0). We have set theta*=0.05.

FIGURE 3. Part 2 of the interactive system for the basic planetary system simulation. Frames 1 and 2 correspond with the interface for comparisons and others frames with all results from comparisons between student's and accurate models.

the free falling object is under the acceleration due to gravity and dissipative forces (see [12]). All variables were adapted to produce results in mks units. Instructions for steps and options for answers are shown in Table 1. With this concept we propose a learning system able to be relevant for real physics applications. Students have opportunities to code, visualize, simulate and evaluate their answers during the learning process. The education level provided by the system focuses on the remembering and understanding cognitive-domain of the Bloom's Taxonomy (see [13]).

5. Evaluation of the System and Results.

We present the evaluation of the system done by graduate students from the Kanazawa Institute of Technology working within the group of Prof. Dr. Hiroshi Kamada. The visual interactive system was presented in lecture format to the students. Fifteen students were asked to answer a questionnaire including six questions to get their impression about the system. Students chose one

TABLE 1. Instructions for steps and answer options for the projectile motion and free fall models. They refer to instructions to help students understand methods to simulate basic physical phenomena. All instructions for steps and answer options are written in the PROCESSING programming language. They follow the same philosophy as steps 1 to 6 in Figure 2. Users go into a designated programs repository and code their answers to step by step simulate these physical phenomena. Answers options are written in cursive letters.

Step Number - Model Instruction for steps $\Rightarrow$ *Options for answers: a), b) and c)*
1 - Free Fall Same as step 1 in Planetary System model $\Rightarrow$ *Same as step 1 in Planetary System model*
1 - Projectile Motion Same as step 1 in Planetary System model $\Rightarrow$ *Same as step 1 in Planetary System model*
2 - Free Fall Same as step 2 in Planetary System model $\Rightarrow$ *Same as step 2 in Planetary System model*
2 - Projectile Motion Same as step 2 in Planetary System model $\Rightarrow$ *Same as step 2 in Planetary System model*
3 - Free Fall Set the position vector at $(200, 0)$ $\Rightarrow$ *a) position = new PVector*$(200, 0)$*; b) position = new PVector*$(100, 100)$*; c) position = new PVector(0,200);*
3 - Projectile Motion Same as step 3 in Free Fall model $\Rightarrow$ *Same as step 3 in Free Fall model*
4 - Free Fall Simulate a dynamic circle. We have to add the position vector and a velocity vector. We assume constant velocity. The position of the falling circle is given by ellipse(position.x,position.y, 40, 40); Previously we have defined a velocity vector with velocity = new PVector$(0, 1)$, which simulates a displacement in vertical direction $\Rightarrow$ *a) position.dist(velocity); b) position.add(velocity); c) position.dot(velocity);*
4 - Projectile Motion Simulate a dynamic circle. We use a data type called float. We have defined a variable float distance; and set the initial distance to 0. The position of the falling circle is given by ellipse(position.x,position.y, 20, 20). Previously we have defined an independent variable distance+, which increases the value of distance. We have set distance+ = 4.0; Please set the x position of the circle to distance+. This will creat a motion effect $\Rightarrow$ *a) position.x = distance+; b) position.y = distance+; c) position.x = distance;*
5 - Free Fall Create a bouncing ball simulation in vertical direction with no dissipative effects. Please set a bouncing ball simulation when ball reaches the floor, that is, if location.x $\geq$ height is, if location.x $\geq$ width $\Rightarrow$ *a) velocity.y = velocity.y* -1; b) velocity.y = velocity.y* -0.5; c) velocity.y = velocity.y;*
5 - Projectile Motion Create a projectile motion in two dimension. We simulate a linear displacement with time+ = 2.2 which increases the value of time. Please set the x and y positions of the circle to time+. This will make a 2D linear displacement $\Rightarrow$ *a) position.x = time; position.y = time; b) position.x = time; position.y = height $-$ time; c) position.x = height $-$ time; position.y = time;*
6 - Free Fall Simulate a free falling object with a bouncing effect and dissipative effects of 0.6 when ball reaches the floor. We have included a gravity vector which is added to the velocity vector $\Rightarrow$ *a) velocity.y = velocity.x* -0.3; b) velocity.y = velocity.x* -0.6; c) velocity.y = velocity.x* $+ 0.6$;*
6 - Projectile Motion Create a projectile motion under acceleration due to gravity. We have set velocity to 8, angle to 45, gravity to 9.81 and time+ to 0.65. Find the correct way to define the horizontal and vertical positions of the projectile motion $\Rightarrow$ *a) position.x = velocity*time; position.y = velocity*time $-$ 0.5*gravity*time*time; b) position.x = velocity*cos(angle)*time; position.y = velocity*sin(angle)*time $-$ 0.5*gravity*time*time; c) position.x = velocity*sin(angle)*time; position.y = velocity*cos(angle)*time $-$ 0.5*gravity*time*time;*

answer from five options to answer each question. All answers included the following options: Completely Agree, Agree, Neutral, Disagree, and Completely Disagree. These questions Q are listed as follows: 1) Students have the opportunity to code the program; 2) Students need advanced computer skill to execute (run) the programs; 3) Students learn how to produce movement simulation; 4) The destination of links was clear; 5) The screen design was clean and easy understandable; 6) Reduce the students' need to look for external information.

Results are shown in Table 2, where data correspond with number of students per selected option and average grades. We assigned grade 5 to the agree completely output, grade 4 to the agree output, grade 3 to the neutral output, grade 2 to the disagree output and grade 1 to the disagree completely output. We found that students agree they have the opportunity to code the program and learn how to create a simulation, destination of links is clear and screen design is clean and easy understandable. Regarding question 2, we found a neutral grade about computing needs of students to execute the program. Answers for question 6 also show a neutral grade about the students' need to look for external information.

Table 2. Results from the evaluation of the system. Number of students per selected option and average grades.

	Agree C. (5)	Agree (4)	Neutral (3)	Disagree (2)	Disagree C. (1)	Result
$Q1$	4	6	5	0	0	3.9 'Agree'
$Q2$	2	4	3	5	1	3.1 'Neutral'
$Q3$	4	7	3	1	0	3.9 'Agree'
$Q4$	2	7	4	2	0	3.6 'Agree'
$Q5$	3	8	3	1	0	3.9 'Agree'
$Q6$	1	4	9	1	0	3.3 'Neutral'

As described before the system was presented in a conference format to the students. As future work we propose that they make a hands-on evaluation of the system to get a first-hand evaluation. As can be seen in Table 2 some students think in a neutral way that the system reduces the students' need to look for external information. We believe that these neutral scores will be improved if students make a hands-on test of the system instead a first impression gotten from a speech presentation.

6. Conclusions.

In this system, most of the programming work is based on instructions established in advance by authors to make students create their simulation without difficulties. We present a system concept to point students how to code a simulation and confront their answers with accurate solutions afterwards. This is, the characteristics of the system include the ability to teach programming techniques, identify user errors, point where the user made a mistake and suggest code inputs in the user's scripts. We have presented two examples of simulations including real physical phenomena, which are based on the equations of kinematics. All simulations and comparison programs work fine, we are now focusing on creating a full integrated system considering that the availability of a friendly and comprehensive interface is important to make the interactive system motivating and easy to understand for users. Furthermore, we also are working on creating a bigger variety of physics exercises to make a diverse system.

Acknowledgment. This work was supported by the Japan International Cooperation Agency. J. J. Jiménez-Torres thanks to the Kanazawa Institute of Technology where this work was done.

REFERENCES

[1] United Nations, *Economic and Social Council, Dialogues at the Economic and Social Council*, http://www.un.org, 2011.

[2] United Nations Educational, Scientific and Cultural Organization, *Education Strategy 2014-2021*, UNESCO Open Access Repository, www.unesco.org, 2014.

[3] C. E. Leiserson and C. Vinney, Science professors need leadership training, *Nature*, vol.523, 2015.

[4] F. K. Hwang and F. Esquembre, Easy Java simulations: An interactive science learning tool, *Interactive Multimedia Electronic Journal of Computer – Enhanced Learning*, vol.5, 2003.

[5] D. Sage and M. Unser, Teaching image-processing programming in Java, *Signal Processing Magazine*, vol.20, no.6, pp.43-52, 2003.

[6] S. Deb, Search for effective distance learning in developing countries using multimedia technology, *Signal Processing and Multimedia, CCIS*, vol.123, pp.253-259, 2010.

[7] P. X. Contla, H. Kamada and D. Takago, Visual interactive learning system using image processing about multiple programming languages, *Proc. of 2016 IEEE the 5th Global Conference on Consumer Electronics (GCCE)*, pp.448-452, 2016.

[8] H. Kamada, K. Nishikawa and Y. Okui, The visual interactive programing learning system using image processing, *Proc. of the 3rd International Conference on Computing Measurement Control and Sensor Network*, pp.158-161, 2016.

[9] Y. Sabinas, H. Kamada and D. Takago, Interactive education system about image processing, *Proc. of the 13th International Congress on Innovation and Technology Development*, pp.1-3, 2016.

[10] D. Shiffman, *The Nature of Code: Simulating Natural Systems with Processing*, 2012.

[11] C. Reas and B. Fry, *Processing: A Programming Handbook for Visual Designers and Artists*, Massachusetts Institute of Technology, 2014.

[12] H. Benson, *University Physics*, Wiley, 1996.

[13] L. W. Anderson, D. R. Krathwohl and B. S. Bloom, *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, New York, Longman, 2001.