Simulation as an education tool

Tennó DAIKI
Department of Media and Education Informatics, Eötvös Loránd University
Budapest, H-1117, Hungary

and

Péter SZLÁVI
Department of Media and Education Informatics, Eötvös Loránd University
Budapest, H-1117, Hungary

and

László ZSAKÓ
Department of Media and Education Informatics, Eötvös Loránd University
Budapest, H-1117, Hungary

ABSTRACT

Western-style scientific methods put a lot of emphasis on the comprehension and theoretical explanation of phenomena, that is, on the accurate modelling of factors that govern system operations. There are a great number of phenomena which are difficult even for well-equipped specialists to observe directly. Our paper, on the one hand, will present the role of informatics in these fields; on the other hand, we will offer a possible methodological structure that can be used both in the classes of informatics (programming) and in the education of the specific field.

In our view, developing programs for the purposes of simulation is an excellent task in the education of programming, because, for one, it is motivating for the students and, for two, it is possible to introduce each linguistic tool to the extent that the students can not only create but also use the program. In our paper we will demonstrate how a few-line-long program can be used to model phenomena taking place inside a container of gas particles and how specific effects like acceleration or force field can be considered.

Keywords: Simulation, Modelling, Physics, Biology, Chemistry, Programming, objects, Firemonkey.

1. INTRODUCTION

First of all, we would like to point to the essential role simulation plays in the process of learning and scientific exploration. With the progress of informatics, simulation models are offering solutions for fields where no other tool can be utilized or no direct observation is possible. The list of specific examples to support this with is endless; now let us only consider the process of mass structure change, of which we know that it is based on the correlation and collective behavior of its basic elements (like atoms and molecules). Obviously, we cannot observe the phenomenon of, say, moisture condensation on the atomic level, but we are well aware that what we see in our real world, like moisture, rain, and dew, among others, is formed out of the relations of the basic elements.

Simulation tools concentrate on defining basic rules between the basic elements/particles and simplifying/refining this rule system until the result coming from running the model is similar to what we observe in the real world.

With the help of informatics, the processes can be slowed down (like with nuclear fission), speeded up (like with social phenomena), magnified (like with gas molecules), or reduced (like with galactic motions). [1]

The most professional simulation systems are the fruits of gigantic efforts from developers; therefore, they are very expensive. [2, 3] In our paper, we are attempting to motivate our readers to create their own experimentation tools or use existing systems, which can demonstrate the correspondences of the given discipline in adequate detail.

At the end of our general introduction, we again emphasize that simulation is a universal tool that offers the only solution in several contexts. [4]

We would like to address another important aspect regarding simulation. In our experiences, simulation systems are highly interesting for the age group we want to start teaching programming to. Occasionally, it is even too interesting. Note that the most important target group of the IT game industry is this age group, and the majority of the games are interactive simulation systems with professional visual design.

If we accept these as facts, it becomes self-evident that the goal of our programming classes should be to make game-like (simulation) programs, because then the motivation is guaranteed. If we manage to put our simulation models in a practical order, through refinement and development, the students will feel they are progressing on their own while experiencing success in every step of the process of developing more and more advanced games (simulation programs). [4]
In our article, just like in our ongoing research/development project aimed at modernizing simulation tool systems (*SziMOT*-

Projekt), we are offering a curriculum that defines how to incorporate simulation (programming) tasks in the general learning process of programming.

In the following, we will demonstrate this duality in a specific example.

2. DEFINING THE TASK

We can often find similar models that belong to different systems. We assume that if we can model the life of a rabbit population, then we can model the life of a mouse population in the same way.

Perhaps it is also easy to accept that gas particles in a closed space and flies locked in a confined container have similar distributions, even though these two groups are rather different. (On the long run, it will be true that both the flies and the gas particles will have a statistically even distribution in the space available.)

Finally, would we think that there is a parallel between electrically charged particles, changing electrostatic spaces, and social processes?

It is just a matter of presentation. If our students are interested in physical models, then we need to make up a game (simulation) that is connected to physics. If they are more into biology or chemistry, we will turn to those disciplines. While, if it is more humanities that excites them, let us adjust to that. In each case, there are several models that can be built on each other if we use some simple rules.

Introductory example:
Let us consider a habitat and fill it up with several entities of the same kind. (Later on we will define the concept of entity.)
For simplicity's sake, the habitat will be a rectangular space, which is divided into square cells, occupied either by one entity or by none. In the beginning, we do not assume any correlation between the entities; we simply step them into the next neighboring cell if available. [5]

A possible solution to this problem (figure 1: Fizika_01.dpr) is illustrated below, with the initial case and the situation after a long running time.

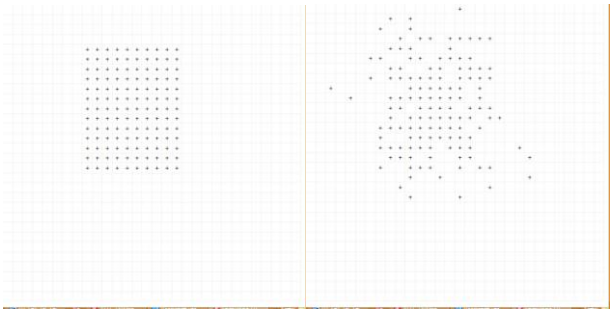

Figure 1: Units arranged in a rectangular shape at start (left), their distribution after 4,000 steps (right).

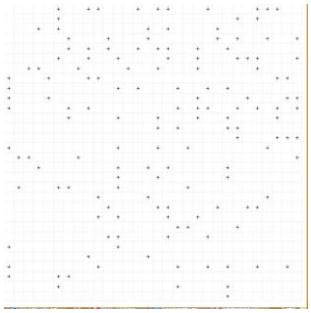

Figure 2: After 108,000 steps their homogeneous distribution becomes visible.

We wish to mention that the program to create these figures does not exceed 30 (yes, thirty) lines of code. (RAD Studio XE5 Delphi was used.)

Our program contains declaration, initialization, conditions, loops, and input/output elements; that is, everything a beginner programmer needs.

If we are not alright with the notion of entity (and we hope this is the case), then we can call them gas molecules and label the simulation as the model of filling up a dish. Once students understand this, we can move on to measuring / examining / displaying the behavior of the system:

- checking distribution in larger units of the space (Fizika_02.dpr), and
- checking collisions on the walls (Fizika_03.dpr).

Once our students reach this point, they will immediately want more of the program. What can we do for them? Let us show them some ways of extension.

- Let us warm a part of the dish to see how our system changes. (Fizika_04.dpr)
- Let us define some correlation between our entities (liquid particles). (Fizika_05.dpr)
- Let us determine the motion of the particles (constant force, acceleration, location-based acceleration, and so on). (Fizika_06.dpr)
- Let us allow entry to and exit from the system (by making the motion of flowing particles constant).
- Let us introduce more kinds of entities into our system.
- Let us add some obstacles.
- Let us examine the collisions on the obstacles.

It is perhaps clear that our simple model was developed so much, through these extensions, that it became fitting for the simulation of how flowing objects behave in moving liquid.
If our playful students are still not satisfied, we can introduce interactive elements into our system, like:

- for moving the particles into one specific part of the space, or
- for minimizing the resistance of the objects in our space.

Since they do not exceed basic programming competences and are built on one another, the above tasks provide students with the sense of success in every step of the process. [6]

In the system of Fizika_06, the maximum step size is 5. We did not pay attention to the formation of drops. We set the horizontal forces like this: left 15%, stays firm 28%, and right 57%; and the vertical forces like that: up 59%, stays firm 28%, and down 57%. After this we ran the program (for a couple of minutes, appr. 17,000 steps). The process in pictures:

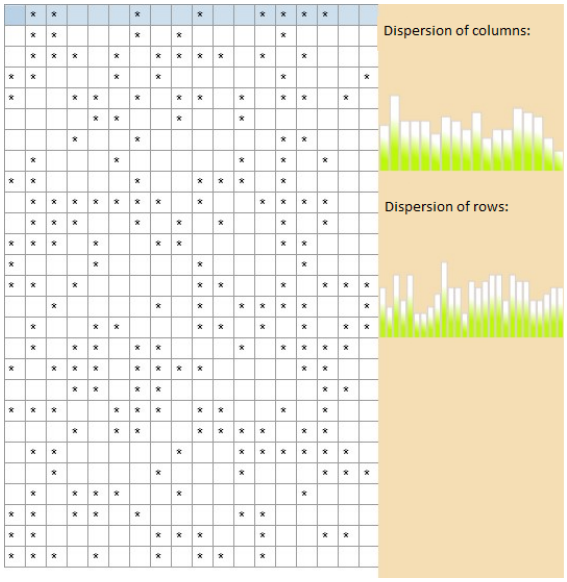

Figure 3: Initial distribution (balanced distribution).

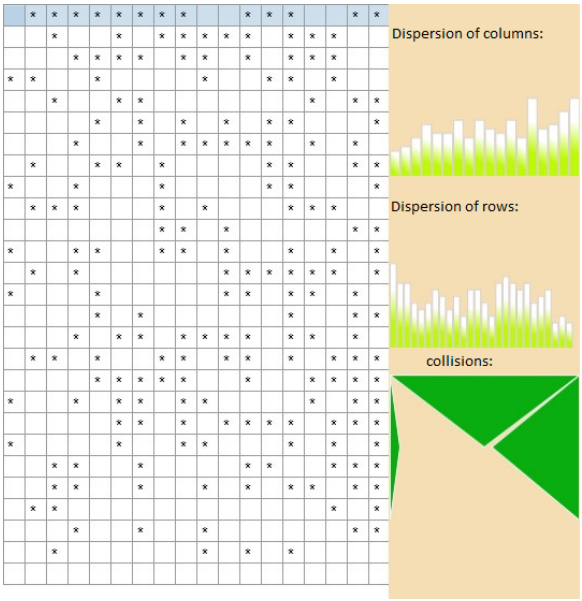

Figure 4: After 3,500 steps (the balanced distribution starts to disappear, but the collisions still reflect the direction of the original movements).

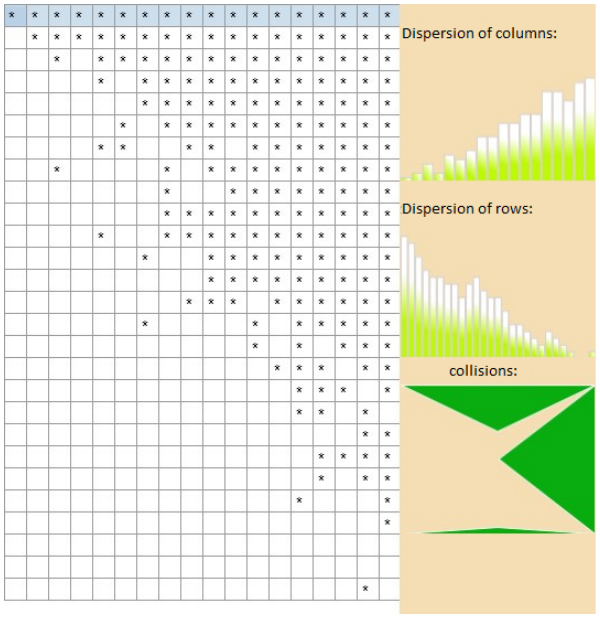

Figure 5: After 17,500 steps. (A right shift in the center of gravity is clearly visible. If we observe the system on the long run, even the leftward movements come through, but their frequency follows force impacts.)

3. DESCRIBING THE PROGRAM VERSIONS

In this chapter of the article, we will present our simulation system that not only engages our students to do programming but it also guides them into reaching more and more complex stages.

Fizika_01.dpr

This is our first operating model, the basic version. We were restricting ourselves to use the simplest tools so to show that the basic algorithms of the simulation are very simple; thus, the task can be solved easily.

As a small amendment, we have added the graphical save function, so we can keep track of the changes of the simulation space and display it later on like a film.

Let us see what we have used in the program.

- To display the simulation space, we have a table called StringGrid. The entities are signaled with '*'. If we add a new box, an entity may appear there during simulation.
- To execute the consecutive steps of the simulation process, we have a Timer component, which initiates the simulation algorithm, based on the constant frame model [1], every 0.001 second.
- To control the simulation, we have a CheckBox (a logical signaling box).

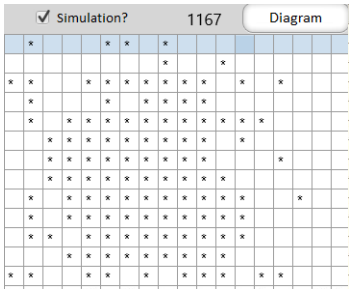

Figure 6: The simulation space and control panel with the button to display the graphic result.

While writing the program, our students may get acquainted with the basics of programming, but we do not explain every single line to them. Based on experience, students (even more than adults) are able to make sense of things and adjust to the environment (so we do not go into details about notions, for example, like object-orientation or event control).
The programming structures students need to know:

- declaration, initialization, conditions, loops

Fizika_02.dpr
We have extended the first program so it can display the distribution of the entities in the simulation space.
Consciously, we have moved forward quite a bit with it, so we can reach the graphic elements as soon as possible. If you think it is a bold decision, you can choose the numeric display of row/column distribution frequency, instead of the graphic display of numeric values.

We have added the option of saving even the graphs this time.

New elements:

- New window to display the results (open a window from a window).
- Button to open the window and step back.
- Panel(-like) component in the window, to draw boxes according to the entity frequency of the given rows and columns. (The long-term even distribution can be detected by its shape.)

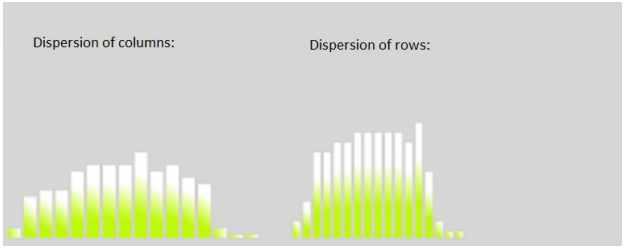

Figure 7: A momentary situation of the distribution. Few molecules appear on the sides, and no molecules have reached the lower rows from their original position yet.

New programming structures:

- concept of unit,
- creating/deleting a component.

Fizika_03.dpr
The previous version is supplemented with the display of collisions in a given time (500 steps). Our, graphic-oriented, solution for this task was to place isosceles triangles next to the appropriate sides of the rectangular so the height of the triangles signals the collisions at the given side after every series of steps. In this way, we always see the current collisions only, but the saved frames played one after another will show an ultimately even distribution.

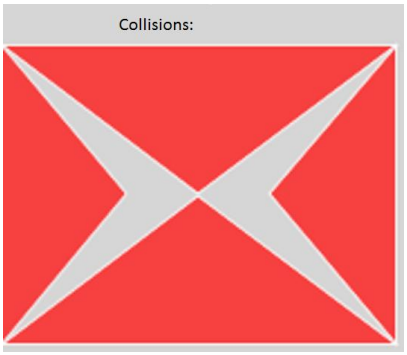

Figure 8: Illustration of collision and pressure on the walls. (Fewer collisions can be observed on the side walls than on the top and the bottom.)

New elements:

- TPath, a free component to display polygons

New programming structures:

- procedures,
- mathematics for calculating triangles.

Fizika_04.dpr
It is in this version that we first apply a tool which enables us to intervene at any point of the simulation. In a certain part of the space, we increase the temperature, that is, we allow the particles of these cells to make bigger (more than one cell) steps. We introduced the parameter "maximum step distance", adding to it a simple surface tool to set the step measure.

We have modified the count of collisions as well, now considering the molecule's change of impulse due to its change of speed as well.

New element:

- New window for setting parameters.

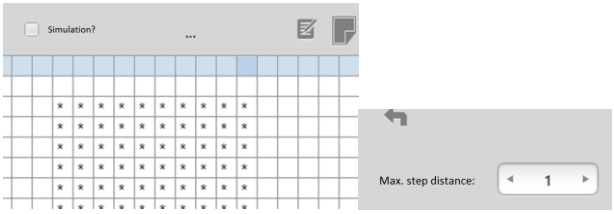

Figure 9: Button for setting parameters (left) and the control panel (right).

- **TSpinBox** for setting the temperature any time during the simulation process.

Fizika_05.dpr
Compared to the previous versions, what we changed here is that from now on the entities are no longer independent but their changes can depend on interactions as well. With this we wanted to show that if the entities attract each other in a certain radius, that will create special groups. (The molecules will form drops.)

Two relevant changes were introduced here. One, the space is randomly filled up at the start; two, the range of attraction (thus, the resulting pattern) can be set during the simulation.
New element:

- Two new parameters, “neighboring radius” and “filling ratio”, were added to the window for setting parameters.

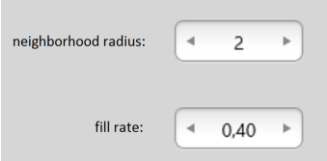

Figure 10: Other elements of the control panel.

- When stepping back from the window, the algorithm randomly places the entities in the given density according to the defined ratio.

Fizika_06.dpr
Our game has been advanced further with a new setting. Now the selected entity is able to decide which neighboring cell to prefer when jumping (direction and distance). We can now model both the vertical and the horizontal forces, either separately or jointly.

To make our input tools more colorful and exciting, we are introducing fairly special control panels in this version. Even if you are not an expert in programming, do not be intimidated by this: we are defining an independent object class, with which we can set the outcomes of our events through a three-stage event system. We would like to show an example even for managing small displays like in the case of mobile phones.

The two trapezes meeting at one of their edges, and the triangle formed between them, cut the space in three. The ratio of the three spaces can be defined and changed easily, with one finger. (Naturally, we know that there are easier ways to do it but in many cases this is still the most effective.)

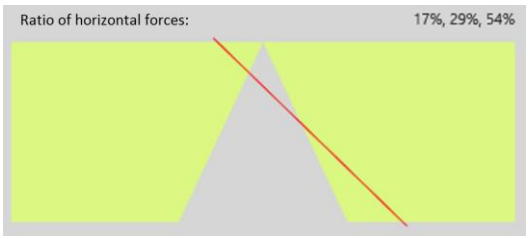

Figure 11: Ratio of horizontal forces: one is 17% likely to step left, 29% to stay firm and 54% to step right.

New element:
- Creating an own object class, used for setting the ratio of three spaces on a touchpad.

4. CONCLUSIONS

With this short series, we have demonstrated a possible way to develop programming skills through simulation tasks.

We are aware that after the initial steps we have jumped forward quite a bit, but we firmly believe that after downloading and examining the source codes, the article will make sense.

It is not a coincidence that many programming competitions (Nemes Tihamér OITV [7], Izsák Imre Gyula science competition [8]), and program product competitions (Neumann János International Talent Competition [9]) assign more and more simulation tasks.

Next to the professional simulation programs, we can now salute programming languages and environments for simulation, specifically designed for high school students. One such is NetLogo [10].

9. REFERENCES

[1] L. Horváth, P. Szlávi, L. Zsakó, **Modelling and simulation**, Mikrológia 1., 2005.
[2] **Interactive Science Simulations**. University of Colorado at Boulder, PhET project, 2014. http://phet.colorado.edu/ (accessed 31 October 2014)
[3] **Virtual Amrita Laboratories Universalizing Education**. Amrita Vishwa Vidyapeetham University, 2014. http://amrita.vlab.co.in/ (accessed 31 October 2014)
[4] P. Szlávi, L. Zsakó, **How to apply informatics in public education**, Informatika a Felsőoktatásban’96 – Networkshop’96, Debrecen, 27–30 August, 1996, conference CD, pp. 534–543.
[5] M. Eigen, R. Winkler, **The laws of the game**, Gondolat Könyvkiadó, 1981.
[6] P. Szlávi, L. Zsakó, **IT competences: Modelling the real world**, INFODIDACT 2013, Zamárdi, 21–22 November, 2013, conference CD, pp. 1–17.
[7] **Nemes Tihamér National Competition in Informatics** – Category of programming, 2014. http://tehetseg.inf.elte.hu/nemes/index.html (accessed 31 October 2014)
[8] **Izsák Imre Gyula science competition**. Zrínyi Miklós Gimnázium, Zalaegerszeg, 2014. http://www.zmgzeg.sulinet.hu/izsak/ (accessed 31 October 2014)
[9] **Neumann János International Talent Competition**, I. Béla Gimnázium, Szekszárd, 2014. http://www.ibela.hu/neumann/ (accessed 31 October 2014)
[10] P.Bernát, “Modelling and simulation in education and the NetLogo simulation environment”, **Teaching Mathematics and Computer Science** Vol. 12., No. 2. (2014), pp. 229–240.