Wild Programming – One Unintended Experiment with Inquiry Based Learning

Pavel Boytchev

KIT, Faculty of Mathematics and Informatics, Sofia University, blvd J. Bourchier 5, 1164 Sofia, Bulgaria
boytchev@fmi.uni-sofia.bg

Abstract. This paper describes one unplanned experiment of a 6th grade student writing her first computer program for 3D graphics before learning any programming language. Some intriguing aspects in her program are analyzed, especially the emerging understanding of key concepts like enumeration, naming conventions of variables and symmetry in 3D space. The paper also identifies two main directions of mental processes. The first direction is actively supported by the school. It is based on presenting and using knowledge in a distilled error-free way. The other direction encompasses techniques needed to identify wrong solutions and to find a way to overcome problems and reach a correct solution. This direction in underrepresented in the educational system and as a result, it is left uncultivated. Students are expected to develop such skills by themselves.

Keywords: programming, cultivated education, emerging understanding

1 About Wild and Cultivated Strawberries

Many people like strawberries, especially the ones that are big, juicy and tasty. These are the cultivated strawberries. The wild strawberries are completely different – they are small, plain, but extremely fragrant. Wild strawberries are perfect for making strawberry jam. Almost three hundreds years ago the French person Amédée-François Frézier brought the wild Chilean strawberry *Fragaria chiloensis* in Europe. When hybridized with the North American *Fragaria virginiana*, it gave birth to the modern garden strawberry [1].

Nowadays, some people are surprised that wild strawberries can be eaten. They don't expect that that a wild fruit can be edible. So far they have only tasted cultivated strawberries, properly wrapped and labeled.

It appears that the cultivation of strawberries has a common ground with the cultivation of … people. For centuries learning and teaching are tightly bound to this cultivation. The situation leads to the question whether we have reached the status of believing that this cultivation is inherent to education.

When we give a toy to a child, we just show quickly how it is used. Then the child continues to play with the toy and to explore its functions. This is a kind of "wild learning". The situation in the classroom is much more cultivated. Everything is being thoroughly premeditated and explained. To some extent this attenuates the natural pursuit of wild experimenting. Within the cultivated education students see only the correct way of solving a problem or undertaking a research. They are detached from the wild exploration, where mistakes are the driving force of learning. People learn from their mistakes – mistakes are as educational as non-mistakes [2]. Unfortunately, we want to exclude all mistakes and even chances of mistakes from the learning process.

Let us consider as an example the discipline *Computer Science* and focus on one of its subdisciplines – *Programming*. The education in *Programming*, independent on the programming language being studied, follows a canonical methodology, which leads to a cultivated, but a sterile state. Is it possible for a student to learn something in this way? Yes, it is, this is the "normal" way of learning things and a lot of people learned to use a programming language in exactly this way. The question is whether wild learning is also applicable in this context. What would happen if students are given only a primary explanation and then they are left alone to experiment with the programming? Would it be possible for complex and abstract concepts in *Programming* and *Computer Science* to emerge? If we forget about the canonical mythology and provide educational freedom, would this lead in a natural way to blending elements from different disciplines?

2 The Experiment

The experiment happened in a casual day, while we were engaged with reviewing more than a hundred multimedia projects written by students from 5th to 7th grades. As expected the projects were highly varied. There were PowerPoint presentations, frame-by-frame-hand-drawn video clips accompanied by personal poetry and even a few animations programmed in OpenGL.

A 6th grader saw the projects and became extremely interested. After seeing several multimedia projects, she said curtly: *Why do we not study how to do this at school? Why do we learn only Paint, Excel and Word?* The reply to these rhetoric questions was that school is not the only place where we can learn new things. Then she asked how she could make some cute animation … *not something recorded by a camera, but animation that is entirely computer-generated*.

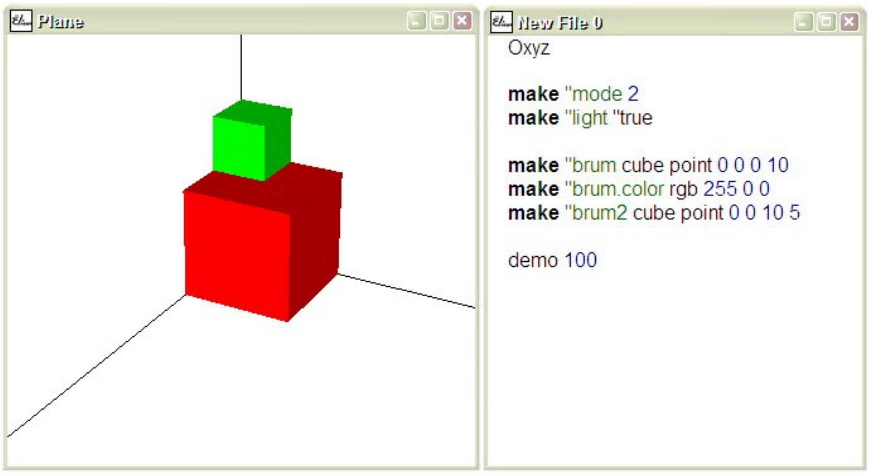
There was a big hesitation whether to tell her about Elica – the programming environment used to build many other educational applications including applications within the frame of three European projects – *DALEST* [3,4], *InnoMathEd* [5] *and Fibonacci* [6]. The main problem was that the girl had never done any programming. She had never written a single command in a programming language, so diving directly into the world of programmed 3D animations could be a disaster. On the other hand, it was a unique moment that she explicitly expressed her strong will to learn something that goes far beyond the school curriculum.

Thus the casual lesson started with some quick introduction to 3D coordinates. The girl was not aware of the Cartesian 3D coordinate system, but she had studied the 2D coordinate system at school. When she was asked *Do you recall 2D coordinates* she answered *Yes*, wrinkling her forehead. It was like just this single question made her step back regretfully. However, we used the two edges of the desk as X and Y axes, and an upright pen as Z axis in order to model a coordinate system. After a moment, while placing hands on desk surface, the girl proudly said that X and Y were forming *a flat plane*.

It was time to move to the next step – introducing coordinates. The girl was shown the approximate positions of objects with coordinates `(10,0,0)` and `(0,0,10)`; and then she was able to point in the space the positions of `(0,10,0)`, `(10,10,0)` and `(10,10,10)`. She was even asked to point `(10,-10,-10)` and after few seconds of hesitation she placed her hand in the correct position in respect to the axes (that was below the desk). It was surprising how fast she managed to get oriented in the 3D space, so it was time to make the final step – writing a true computer program.

For this step we used Elica. Its acronym stands for *Educational Logo Environment for Creative Activities*. Although it is based on Logo, a language largely and wrongly assumed to be childish, Elica provides support for object-oriented, functional and procedural programming – all at the same time. It was quite risky to ask a child that had absolutely no programming experience to write a program. Thus, hoping to make just a "presentation" we showed her a simple program that draws and rotates two cubes. A snapshot of the screen, together with the program code is shown in Fig. 1. The `make` statements define the cubes and their properties, and `demo` is "responsible" for the rotation.

The most surprising element in this program was when the girl was asked to give names to the cubes. She was curious why, but she accepted without problems that all objects in the animation must have their own unique names. In this way she could "touch" the objects and "tell" them what to do. Most likely the problem with naming was that in Paint the picture is not composed of individual entities, but is treated as a single piece of painted nameless strokes.



```
Oxyz

make "mode 2
make "light "true

make "brum cube point 0 0 0 10
make "brum.color rgb 255 0 0
make "brum2 cube point 0 0 10 5

demo 100
```

Fig. 1. The program for creating and rotating two cubes.

Anyway, the girl decided that the cubes must be called `brum` and `brum2` (echoic words corresponding to *whirr* or *buzz*). We did not influence this decision and we did not discuss it with her.

The experiment up to this point was about 5-10 minutes long. The final explanation that we provided was that Elica could use not only cubes, but spheres, cones, and many other shapes. After this note the girl was left along.

3 The Result

Approximately 15 minutes later we went to her room to see what is going on and we were shocked to see a panda on the computer screen, see a snapshot in Fig. 2. This panda was the first program ever of this 6th grader! It was so unbelievingly well done, that we immediately studied it and asked several question:

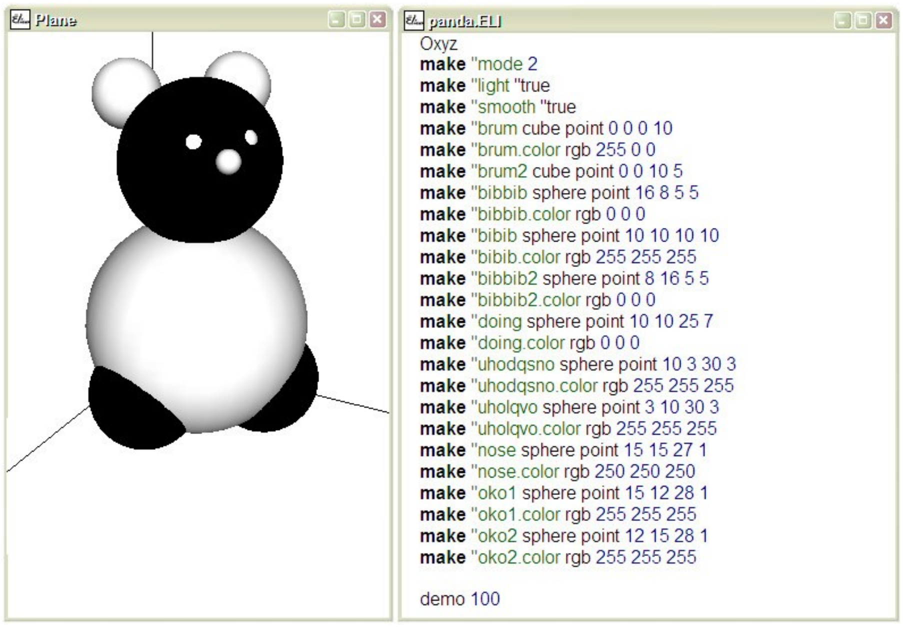
We: *How do you know how to use spheres?*

Girl: *You told me that I can use spheres, so I looked for "сфера" (i.e.* sphere *in Bulgarian) in Google and found that in English it is "sphere". So I just used this word and everything worked so well.*

We: *Did you try other objects?*

Girl: *Yes, but they didn't work out.*

We: *Yes, to construct them you need more numbers, because these objects are more complex.*



```
Oxyz
make "mode 2
make "light "true
make "smooth "true
make "brum cube point 0 0 0 10
make "brum.color rgb 255 0 0
make "brum2 cube point 0 0 10 5
make "bibbib sphere point 16 8 5 5
make "bibbib.color rgb 0 0 0
make "bibib sphere point 10 10 10 10
make "bibib.color rgb 255 255 255
make "bibbib2 sphere point 8 16 5 5
make "bibbib2.color rgb 0 0 0
make "doing sphere point 10 10 25 7
make "doing.color rgb 0 0 0
make "uhodqsno sphere point 10 3 30 3
make "uhodqsno.color rgb 255 255 255
make "uholqvo sphere point 3 10 30 3
make "uholqvo.color rgb 255 255 255
make "nose sphere point 15 15 27 1
make "nose.color rgb 250 250 250
make "oko1 sphere point 15 12 28 1
make "oko1.color rgb 255 255 255
make "oko2 sphere point 12 15 28 1
make "oko2.color rgb 255 255 255

demo 100
```

Fig. 2. A 3D panda – the girl's first program. The long sequence of `make` statements suggests the application of some complex programming concepts.

There were some surprising things in the program. The first objects that the girl added to the cubes had funny meaningless names, like `bibbib` and `doing` (again echoic words). Then she started to embed sense in the names, the panda ears were named `uhodqsno` (right ear) and `uholqvo` (left ear), the nose was called `nose` (in English!)

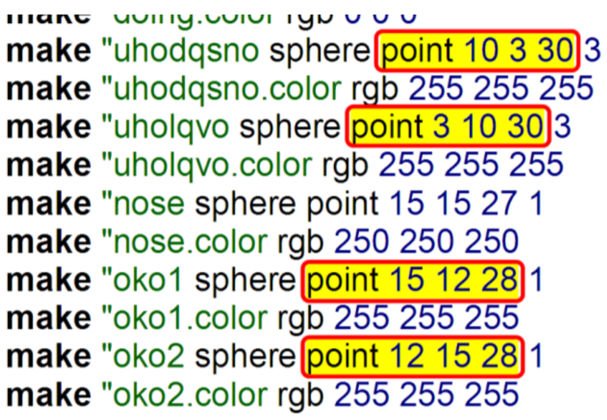
And then suddenly she jumped to a numerical notation, which generates shorter names and is the doorstep to enumeration – `oko1` (eye 1) and `oko2` (eye 2). Enumeration is a key programming concept, which is the core of arrays, cycles and iterations. It is unexpected to observe such transition at so early stage.

Another interesting observation, realized several days later, was the use of symmetry. If we were to make a panda, we would orient it along some of the axis, so that the whole panda body is symmetrical in respect to a trivial vertical plane (like the plane `y=0`). This would make it much easier to position symmetrical body parts like eye, ears and legs. If one part has coordinates `(x,y,z)`, then its symmetrical part would be at `(x,-y,z)`.

However, the girl's panda was not oriented in a way to use such idea, yet it was completely based on symmetry – the symmetry plane was the bisecting plane `x=y`. This plane makes points `(x,y,z)` and `(y,x,z)` symmetrical.

Some of the symmetrical coordinates are shown in Fig. 3. The spheres for the ears (the statements that create variables `uhodqsno` and `uholqvo`) are placed at (10,3,30) and (3,10,30). The centers of the eyes (`oko1` and `oko2`) are at (15,12,28) and (12,15,28).

The 3D objects that the girl created were appended to the definitions of the two cubes. When the panda bear turned the cubes were poking out of her lower back – see Fig. 4. It looked like these leftovers were the first ever programming bug of the girl, but this conclusion was premature and … wrong. The girl explained us that *these cubes are the chair of the panda and that everything is correct!!!*



```
make "uhodqsno sphere point 10 3 30 3
make "uhodqsno.color rgb 255 255 255
make "uholqvo sphere point 3 10 30 3
make "uholqvo.color rgb 255 255 255
make "nose sphere point 15 15 27 1
make "nose.color rgb 250 250 250
make "oko1 sphere point 15 12 28 1
make "oko1.color rgb 255 255 255
make "oko2 sphere point 12 15 28 1
make "oko2.color rgb 255 255 255
```

Fig. 3. Close-up of some symmetrical coordinates.

Later on the same day the girl made another program – a face of a child with lips, eyes with irises, nose and hair. We showed her some simple form of animation like inflating and deflating the face by changing one of its radii. It was quite interesting how the girl “accepted” that a sphere had actually three radii – one along each of the axis; and by making them non-equal we could deform the sphere – and the girl quickly completed the sentence for us – *into an egg*.

4 Afterthoughts

The result of this experiment showed that programming is not hard at all if we do not insist to tell all details and provide complete scientifically correct explanations. A child can start programming without understanding *everything about the program*. This method is much close to the exploration of an unknown toy, when the child is left to experimentally find out what can be done.

Additionally, letting a student play with and in (!) a programming environment does not impose any restrictions to imagination. While creating something entirely by her, the 6th grader freely integrated art activities with programming. If an adult was about to write his/her first program for 3D graphics, he/she would most likely start with something more conventional, more systematic … or even more cultivated (like reading the documentation).

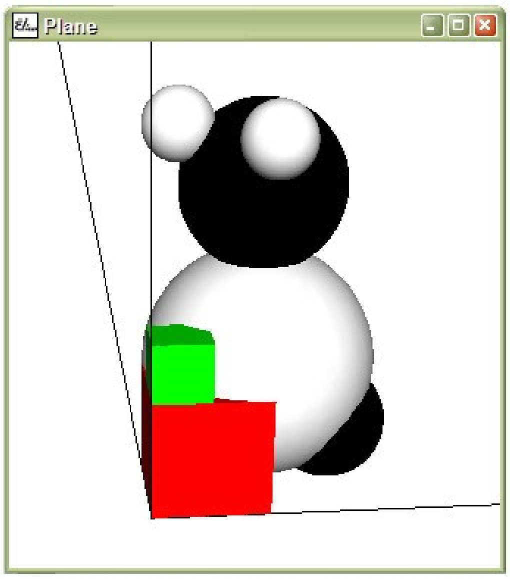


Fig. 4. There is no bug here, but the chair of the panda.

The experiment shows one of the advantages of the programmable educational environments. In such environments students have at their disposal instruments for describing not only what they do, but also the individual steps of their constructions. Students' programs, independent on their complexity or simplicity, are projection of students' thoughts. Even "the most innocent" elements like the selected naming convention of variables, provide clues about the existence of specific skills and the level of understanding of key concepts.

Cognitive psychology explores various types of thinking. Two of the most distinguished types are the *vertical thinking* and the *horizontal (lateral) thinking* [7]. Some of the main features of both thinking types as identified by Paton [8] based on [9] are listed in Table 1. The cultivated approach in education fits perfectly to the vertical thinking, while the wild approach – to the horizontal one.

Table 1. Vertical and horizontal thinking mapped to cultivated and wild education

Feature	Vertical thinking	Horizontal (lateral) thinking
Characteristics	selective, analytical	generative, provocative
Focus on	rightness	richness
Individual steps	must be always correct	some could be wrong
Negative experience	blocks off certain pathways	does not exist
Thinking process	finite	probabilistic

Doing research by writing a computer program reveals much more information if we focus not only on the final program as a static artifact, but also on the program's evolution from scratch till the end, passing through many incomplete and buggy states. This evolution shows a new class of thinking and is indicative for the path of gaining concrete skills and understanding key threshold concepts. The *horizontal thinking* is the one which happens when students stumble upon a wrong solution and try to traverse the solution to a correct solution. This thinking helps the students to "feel" when a research is going in the wrong direction before it is too late. This is the thinking that allows the students to attempt different solving strategies over a problem instead of being blocked off by failures.

Educational environments that allow experimentation via programming develop not only the vertical, but also the horizontal thinking. A programming description of a solution is rarely written perfectly from the very beginning. Often it is required to remove bugs or to improve some existing elements. Debugging and optimization are some of the processes that develop horizontal thinking. Unfortunately, horizontal thinking is not taught at school, but is expected to be learned. This shows one visible discrepancy between what is taught and what is expected to be learned. The vertical thinking is completely cultivated up to the level of lack of critical thinking – here is a problem, here is an algorithm for solving it, follow the algorithm and you will get a correct solution. At the same time the horizontal thinking is growing in the wild, uncontrolled and undirected.

Would it be better to restore the balance between both thinking types? Could we make the vertical thinking wilder (i.e. to make it more independent and more creative by deframing students' thinking and letting them experiment)? Or could we make the horizontal thinking at least more cultivated (i.e. to help students to analyze wrong situations and developing skills for searching new solutions)? These are questions that need yet to be answered.

5 As an Epilogue

The experiment described in this paper was not planned, that is why it was not possible to observe the process of the creation of the panda. Only one student was involved, so it is too early to draw general conclusions. It is not known whether the wild programming always leads to small aromatic fruits or the result was pure fortuitous event. Maybe wild programming is not applicable to mass education? Maybe it is more suitable for individual learning? The answers of these questions are unknown, but the thing, which is known is that without the efforts of Amédée-François Frézier, today, three hundreds years later, it would be impossible to enjoy the garden strawberry. And something else is also known. Frézier not only brought the strawberry to Europe, but he was the mathematician, whose works laid the fundaments of the 3D geometry in military construction and engineering.

As for the usage of digital technologies in education, the Logo-philosophy (a main topic in the international conference Constructionism 2010 [10]) is not to focus only on the informational or the technological sides, but to fully explore the potential of students to be constructors of their knowledge, to learn through inquiry and to share their works.

References

1. Darrow, G.: The Strawberry: History, Breeding and Physiology, Chapter 4: The Strawberry From Chile. Holt, Rinehart and Winston, New York (1966)
2. Boytchev, P.: Pedagogical Inversion**,** Presented at the *4th International Conference for Theory and Practice in Education*. Budapest (2011)
3. Boytchev, P., Chehlarova, T., Sendova, E.: Enhancing Spatial Imagination of Young Students by Activities in 3D ELICA Applications. In: 36th Spring Conference of the Union of Bulgarian Mathematicians, pp. 109--119. Varna (2007)
4. DALEST project, http://www.elica.net/site/museum/Dalest/dalest.html
5. InnoMathEd project, http://www.math.uni-augsburg.de/prof/dida/innomath
6. Fibonacci project, http://fibonacci.uni-bayreuth.de/
7. Robertson, S.: Types of thinking. Routledge, London (1999)
8. Paton, B.: Lateral Thinking. http://www.solutioneers.net/solutioneering/lateralthinking.html
9. Bono, E.: Lateral Thinking: A textbook of creativity. Penguin, Harmondsworth (1977)
10. Clayson, J., Kalas I., (eds.): Proc. of Constructionism 2010. Paris, France, 2010. Comenius University, Bratislava (2010)