Constructionism as making construals: first steps with JS-Eden in the classroom

Antony Harfield, *antonyh@nu.ac.th*
Department of Computer Science & Information Technology, Naresuan University, Thailand

Rene Alimisi, *info@edumotiva.eu*
Edumotiva-European Lab for Educational Technologies, Greece

Peter Tomcsányi, *tomcsanyi@fmph.uniba.sk*
Department of Informatics Education, Comenius University, Bratislava, Slovakia

Nick Pope, *nick@dcs.warwick.ac.uk*
Meurig Beynon, *wmb@dcs.warwick.ac.uk*
Department of Computer Science, University of Warwick, UK

Abstract

JS-Eden is an environment for learners to build software artefacts that relies on construction by 'making construals' using observables and dependencies. JS-Eden is proposed as an alternative to procedural or object-oriented constructionist environments. In this paper we present a small experiment in which JS-Eden is introduced to 25 high school students. The observations and feedback suggest that although there are improvements to be made to JS-Eden's user interface for learners, the principles of constructionism by making construals can be readily applied in a classroom for domain learning. Comparisons are drawn with existing constructionist environments, and it is argued that making construals in JS-Eden is a better paradigm for children engaged in the "instructing", "animating" and "modulating" activities that are key in working with digital media.

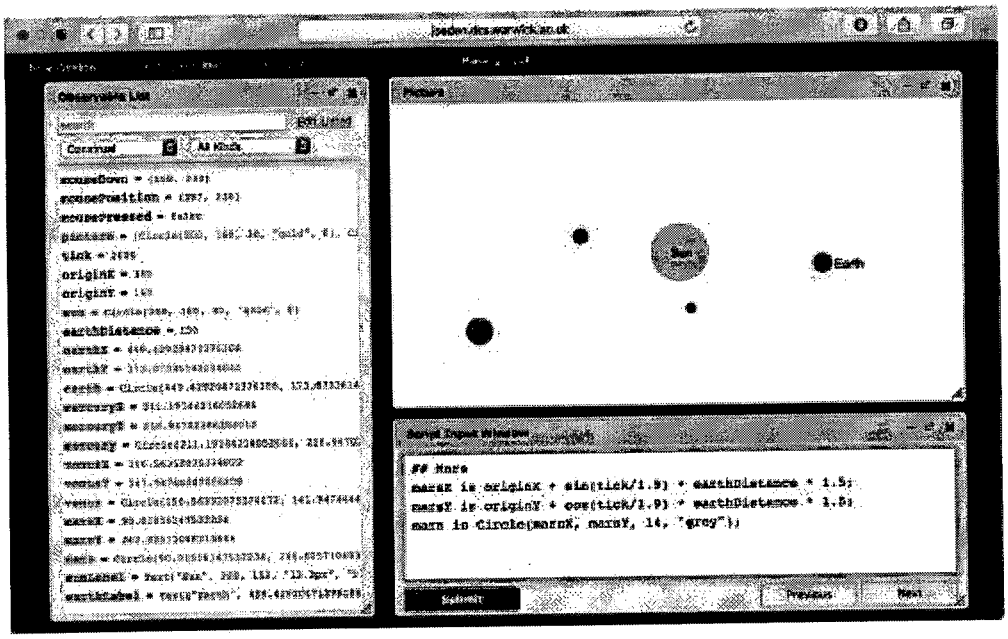


Figure 1. The solar system construal.

Keywords

Constructionism, programming, learning technology, Empirical Modelling, making construals

Introduction

A recurring tenet of Papert's constructionism is that children should not learn programming for the sake of programming. Instead, children can use programming to create contexts in which a wide variety of learning experiences might occur (Papert, 1980). Many of the most popular constructionist programming environments today (e.g. Alice, Greenfoot, Scratch) take a computer scientist's view of programming as procedural or object-oriented (Utting et al., 2010) and, due to the challenging nature of programming, the learning often focuses on the tool or language. While there are advocates of computational thinking as a learning outcome in itself (Wing, 2006), Papert's vision for constructionism was clearly much greater. Brennan contends that, although our classroom technologies have changed, we are still "stuck in a technocentric view" (Brennan, 2015). On the other hand, Ackermann points out that a digital native's concept of 'programming' spans a broader range of activities involving "instructing", "animating" and "modulating" (Ackermann, 2012). Taken together, these critiques suggest that the traditional procedural view of programming is not only unsatisfactory for supporting current learning activities but also insufficient for the vision of construction as a way to learn in any domain. In this paper we propose to support constructionism by 'making construals' – an activity that is more general than conventional programming, and promotes a less technocentric view more closely aligned with the activities that Ackermann identifies. Our claim is that this non-procedural, non-object-oriented approach is more accessible to young people while still supporting the breadth of constructionist activities in which today's digital natives are engaged.

The idea of 'making construals' originates in a reconceptualisation of computing called Empirical Modelling (EM) that is radically different from the paradigm of computational thinking (Beynon, 2012). Rather than focusing on writing programs to achieve specific functional goals, EM puts its primary emphasis on making construals which are built up incrementally and have states and transitions closely connected with an object of study in the learning domain. The characteristic guiding principle of EM is that the learner should be able to directly experience a correspondence between the state of a construal and the state of an external object. The learner works as a modeller, building the construal by creating observables and dependencies. If we consider a learning situation where the object is the motion of planets in the solar system then the observables might be the Sun, the Earth and the position of the Earth in relation to the Sun. Dependencies are connections made between observables, for example how the position of the Earth depends on the position of the Sun and the day of the Earth year.

JS-Eden for making construals

Construals are created by making connections using observables, dependency and agency (Beynon, 2012). JS-Eden is a web-based environment for creating construals that has been developed by researchers at University of Warwick (Beynon et al., 2015). JS-Eden has previously been utilised as a constructionist tool for high school students to build construals for learning mathematics (Harfield and Beynon, 2012). Construals are constructed by entering script-like definitions into JS-Eden which interprets the definitions on the client-side of the web browser. The primary interface for JS-Eden consists of a Script Input Window, an interactive canvas ("Picture") and an observable inspector ("Observable List"). A developer, or teacher, or learner, constructs a construal in a progressive manner, by adding or modifying observables, dependency and agency on-the-fly without recompiling. Unlike many procedural programming environments which have a two stage process of creating and running, JS-Eden is interpreted and constantly "live", representing state as it is in the current moment. JS-Eden can be used to build models from scratch, and to exercise or 'remix' existing models. The focus of this paper is on the use of JS-Eden for building simple models from scratch. Figure 2 depicts the typical starting screen of JS-Eden showing an 'empty' construal.

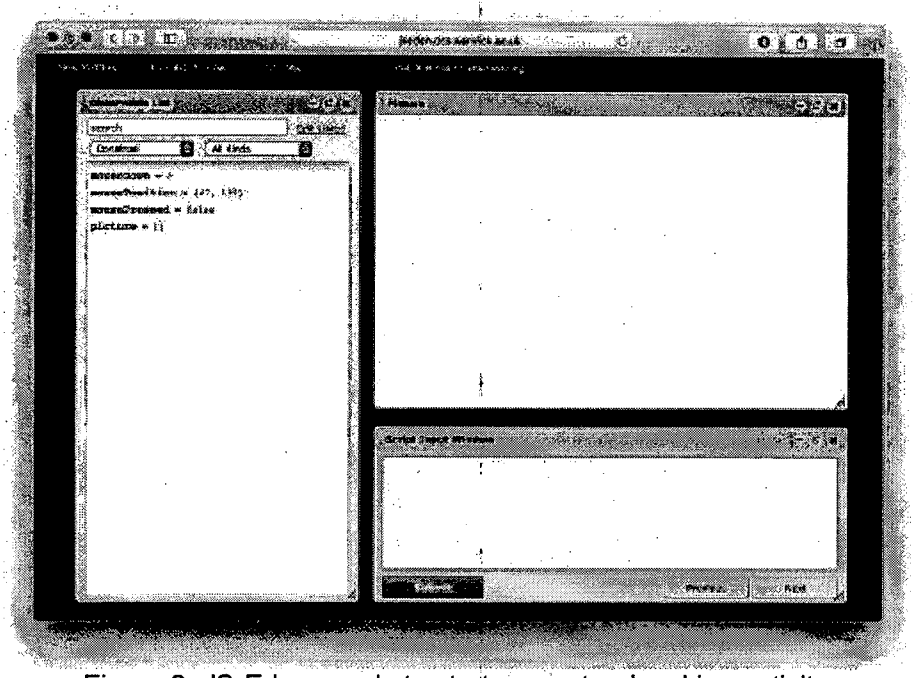


Figure 2. JS-Eden, ready to start a construal making activity.

Early JS-Eden trial in school

CONSTRUIT! is an on-going project to develop online resources for making construals. As part of the project, a workshop was organised in Athens, Greece, of which one part was to test the viability of teaching using JS-Eden in the classroom. The main aim of this experiment was to provide teachers with a model class and to analyse student reactions to the JS-Eden environment and the learning material. Furthermore, the experiment was a way to test the claim of this paper that JS-Eden is an accessible environment for digital natives that supports a breadth of constructionist activities.

The workshop was held at 2nd Experimental Lyceum of Athens, Greece, for 3 teaching periods (approximately 2.5 hours) on Tuesday 22nd September 2015. The school was chosen because the teachers and students have experience with trying new approaches and technologies. The participants were 15-16 year old students. There were 25 students in total, and they mostly worked in pairs. Two teachers and a number of assistants and observers from the CONSTRUIT! Team were present during the session.

The students had no prior experience of JS-Eden. The workshop was designed to introduce the basics of making construals in JS-Eden and help the students become familiar with the kind of things they could build with JS-Eden. The aim of the workshop was not to teach programming or computational thinking. At the beginning of the workshop, no more than 5 minutes were spent explaining the 3 main components of JS-Eden (the Script Input Window, Picture, and Observable List as shown in Figure 2) with a short example. The students were then given a worksheet that explained what to do next and walked them through the basic building blocks of the environment. Each topic involved at least 1 question. In total there were 11 tasks in the worksheet.

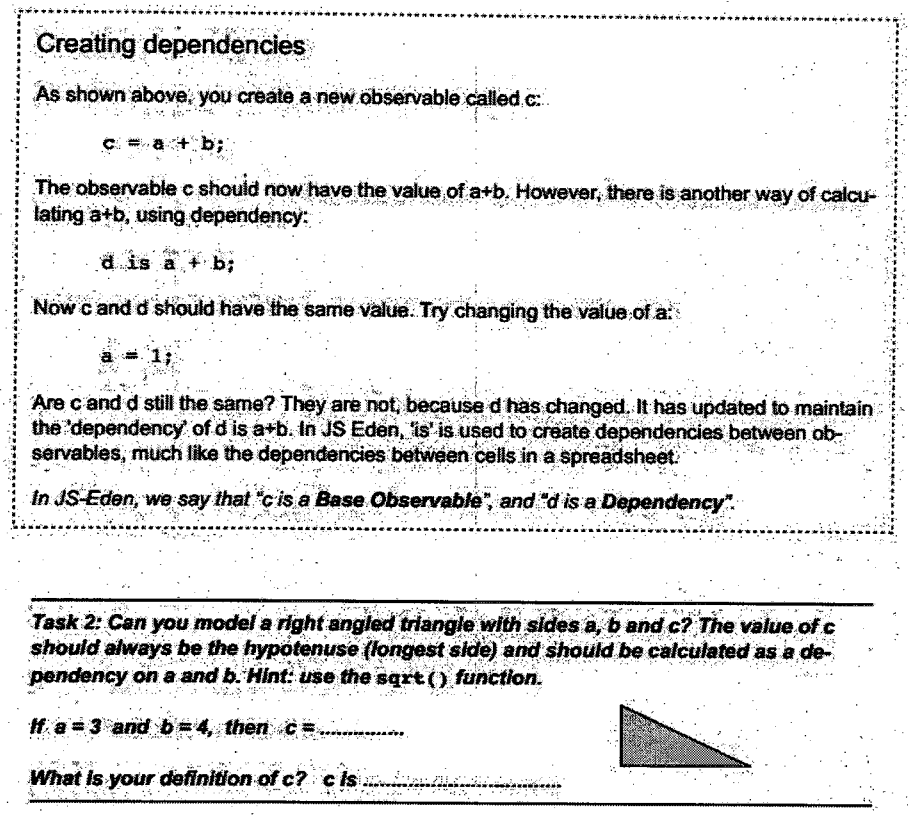
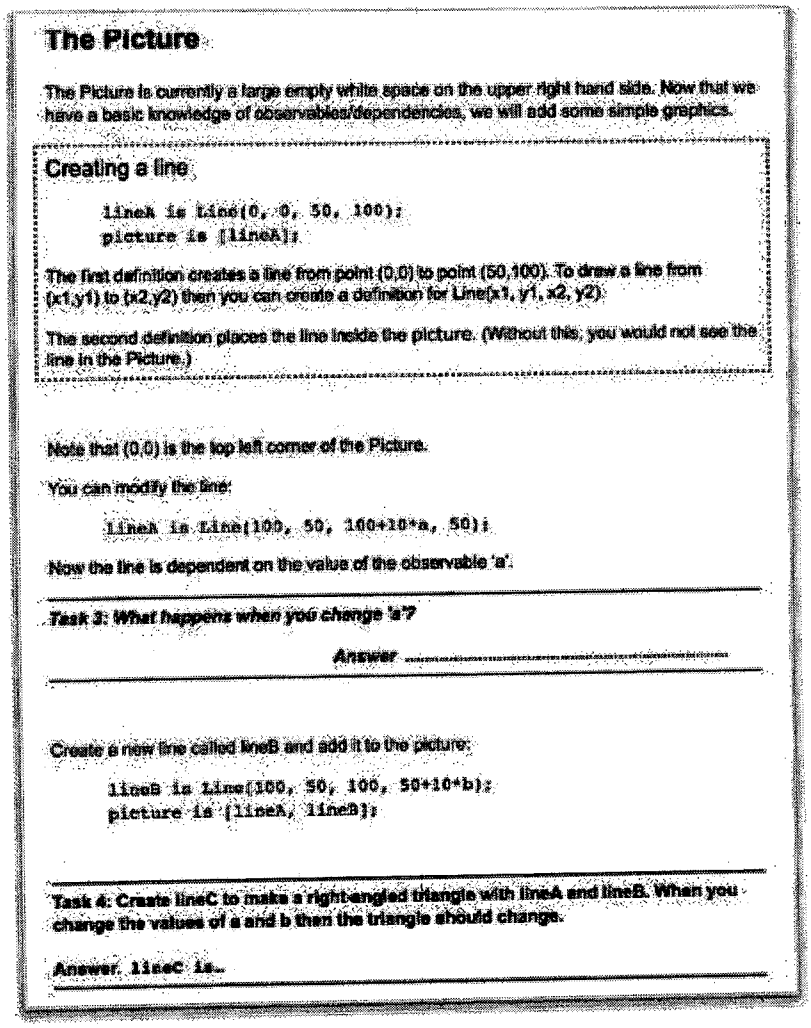


Figure 3. Introduction to dependencies and the Pythagoras's theorem task.

Figure 3 shows the part of the worksheet where the notion of dependency was explained for the first time (the part framed by dashed line). Some of the observers were able to see the "Aha! effect" in some groups of students when they observed that the value of observable d changed after changing the value of observable a. This is an important concept for the students to grasp as it is unrelated to procedural programming. The immediately following Task 2 was oriented not only to practice the new knowledge of dependency, but also to use some of their existing mathematical pre-knowledge in a new situation. Figure 4 shows the next part of the worksheet which introduces the "picture" observable and the Line graphic object. Tasks 3 and 4 concentrate on both creating graphics in JS-Eden and the use of dependencies. They both also relate to drawing a triangle that illustrates Pythagoras's theorem featured in Task 2. Later in this paper we will discuss the outcomes of Tasks 2-4 in more detail.

Figure 5 shows an example of one of the later tasks. This relates to building a model of the solar system, where the students must create dependencies that position the Earth dependent on the day of the year.



The Picture

The Picture is currently a large empty white space on the upper right hand side. Now that we have a basic knowledge of observables/dependencies, we will add some simple graphics.

Creating a line

```
lineA is Line(0, 0, 50, 100);
picture is [lineA];
```

The first definition creates a line from point (0,0) to point (50,100). To draw a line from (x1,y1) to (x2,y2) then you can create a definition for Line(x1, y1, x2, y2).

The second definition places the line inside the picture. (Without this, you would not see the line in the Picture.)

Note that (0,0) is the top left corner of the Picture.

You can modify the line:

```
lineA is Line(100, 50, 100+10*a, 50);
```

Now the line is dependent on the value of the observable 'a'.

Task 3: What happens when you change 'a'?

Answer .....

Create a new line called lineB and add it to the picture:

```
lineB is Line(100, 50, 100, 50+10*b);
picture is [lineA, lineB];
```

Task 4: Create lineC to make a right-angled triangle with lineA and lineB. When you change the values of a and b then the triangle should change.

Answer. lineC is..

Figure 4. Introduction to graphics and using the knowledge about dependencies in a new context.

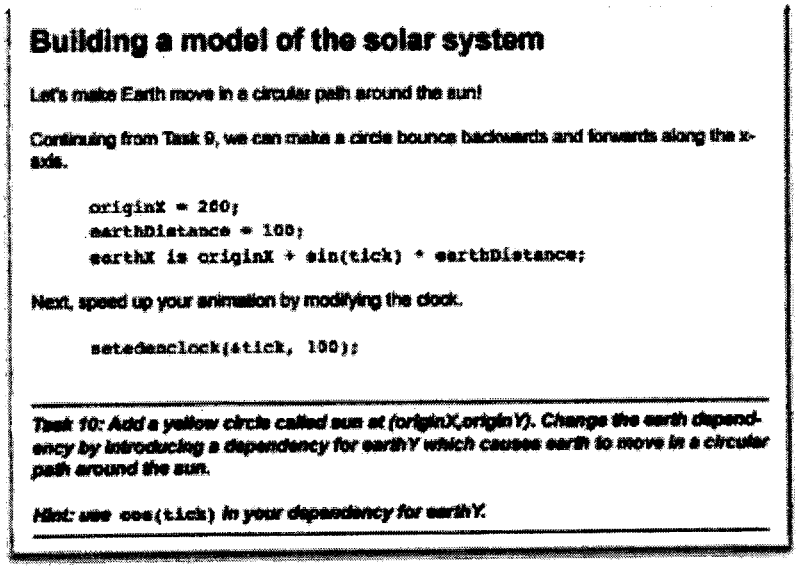


Figure 5. Later task on building a model of the planets in the solar system.

By the end of the workshop the students could have created a fully animated solar system with 4 planets moving around the sun at their correct relative velocities. The status of the construal after completing all the tasks in the workshop is shown in Figure 1. The three definitions in the Script Input Window represent the dependencies required to create a circle for Mars and animate it so that it moves around the Sun in a circular path. The first 2 lines define the x and y position of Mars, which are dependent on the position of the Sun (*originX* and *originY*), the day of the Mars year (based on a factor of *tick*) and the distance from the Sun to Mars (approximated as *earthDistance* * *1.5*).

Results of the trial

During the session, the students' interactions with the environment were recorded by capturing their submitted inputs in the background. The recordings of student interactions showed that all of the student groups reached task 8 out of a total 11 tasks. The last 3 tasks related to building the solar system model, and while more than 75% of the students made some progress on these tasks, only one group was able to complete the final task (which was purposefully difficult and open-ended to push the students who found the earlier material easy). The later tasks involved the use of sine and cosine which had not been taught as part of their regular classes which may have slowed the progress of some students. However, the assistants were active in helping students through the difficult exercises which contributed to the high completion levels.

Examples of the struggles and breakthroughs that the students experienced can be found in the recordings. Consider the tasks 2 (shown on Figure 3), 3 and 4 (both shown on Figure 4) which involved drawing 3 lines to make a triangle and required the students to apply Pythagoras' theorem in order to complete the tasks. Figure 6 shows three extracts of the recordings relating to tasks 2-4: the script entered by the students is displayed in bold and the comment above shows the timestamp of JS-Eden session (e.g. *26m 16s* is 26 minutes and 16 seconds into the session). Note that it is not evident from the scripts whether the students received any assistance during these periods – hence their breakthroughs could be the product of an individual, the group, intervention from an assistant, or a combination of all three.

In the case of Group A, after 26 minutes the students seem to be considering how to use the square root function (sqrt) to solve task 2 which requires them to calculate the hypotenuse of a

right angled triangle using Pythagoras' theorem ($a^2 + b^2 = c^2$). After 30 minutes they enter their definition of c as sqrt(a*a) + sqrt(b*b). This is a valid JS-Eden definition, but mathematically it is wrong. After a further 3 minutes they have realised their mistake and entered a new definition for c (perhaps through their observation of the values of a, b and c in the Observable List). *Group A make a mathematical error that they are able to observe and correct.*

The observations of Group B start with them making some mistakes about how to make a definition. They might be new to programming as they struggle with the equals operator between the 26th and 28th minutes. They make a further error in the 30th minute, but their understanding of Pythagoras's theorem is correct. It takes a further 2 minutes for them to refactor to obtain a syntactically correct definition for c (although it uses "c =..." instead of "c is..." which means there is no dependency). *Group B have a correct mathematical understanding, but struggle to represent it in the language of JS-Eden.*

In contrast to Group B, Group C obtain a syntactically and mathematically correct definition of c on their first attempt at task 2. However, in task 4 (after 29 minutes) they are initially unable to apply their knowledge to create the line of the triangle representing the hypotenuse. After 31 minutes, Group C is trying to use the sqrt function to create the line that connects LineA and LineB (mixing up the mathematics of triangles with the drawing of triangles). At 41 minutes they have mastered the syntax for drawing the line and finally at 49 minutes they find a definition that satisfies the requirements to create the triangle. *Group C have a misconception about how to draw the lines of a triangle, but through experimentation they are able to make sense of the problem and solve it.*

This small sample of extracts demonstrates how the students overcame three struggles: a misunderstanding of Pythagoras's theorem, a misrepresentation of Pythagoras's theorem in JS-Eden and a misconception about how Pythagoras's theorem relates to drawing triangles in JS-Eden.

Group A

26m 16s
a = sqrt(9);

26m 43s
b = sqrt(16);

30m 31s
c is sqrt(a*a) + sqrt(b*b);

33m 17s
c is sqrt(a*a+b*b);

Group B

24m 33s
a = 3;

24m 45s
b = 4;

26m 46s (error)
c = c * c;

Group C

15m 12s
c is sqrt(a^2 + b^2) ;

...

29m 8s
LineA is Line(100, 50, 100+10*a, 50);

31m 36s
LineC is sqrt(LineA^2 + LineB^2) ;

32m 28s
LineC is sqrt (LineA^2 + LineB^2);

32m 44s
LineC is sqrt(LineA^2 + LineB^2) ;

37m 51s
LineC is Line(100, 50, 100, 130) ;

40m 29s
LineC is Line(100, 50, 130, 100);

41m 39s

## 27m 24s **c = a+b;** ## 28m 3s (error) **c*c=;** ## 30m 58s (error) **c*c=a*a+b*b;** ## 32m 17s **c = sqrt(a * a + b * b);**	**LineC is Line(100, 50, 100+10*a, 50);** ## 43m 52s **LineC is Line(100 , 50+10*b)^2 + (100+10*a , 50)^2;** ## ... ## 49m 58s **LineC is Line(100, 50+10*b, 100+10*a, 50);**

Figure 6. Three interaction extracts that demonstrate the difficulty faced by the students in applying Pythagoras' theorem.

Immediately after the session, the teacher was interviewed, specifically to understand her reaction to the activities undertaken by the students. The teacher expressed the view that the students reacted well to the activity, which was evident from the fact that their attention was undisturbed for 3 periods when usually they would have taken breaks. The teacher said that they would like to continue using this tool in their classes. They observed that it is different style of 'programming' to Scratch and Logo (with which they have some experience) because it has more reliance on typing and entering definitions. For this reason, they felt it was closer to traditional programming that requires textual input. The example given by teacher was that it is similar to Prolog in the way that it is an interpreted, definition-based language.

Feedback was also obtained from the CONSTRUIT! project members who were present during the session. They highlighted problems with the JS-Eden environment which mostly focussed on the difficulties that students had with the lack of suitable error messages and poor feedback. This is possibly an indication that the problems were not conceptual, but more a consequence of JS-Eden's migration from being an expert's tool to a learner's tool. The observers noticed that the students appeared to be experimenting with the behaviour of the environment, either in a focused way, or by random 'prodding' to understand how it works.

Several of the observers noted that the tasks involved the students applying their basic mathematical knowledge (e.g. modulo arithmetic, Pythagoras's theorem, and sine/cosine). According to observers, some students were not aware or could not access this knowledge. Were the students unaware of the required mathematical knowledge? This might be a possible explanation – although the class teacher stated that these mathematical topics have been taught and they are integral parts of the school mathematics curriculum. Another possible explanation may be related to the context of the activity – students might have had an expectation that they were programming and therefore they did not treat the exercise as being related to 'doing maths'. In other words, being 'hooked' in their perception of the context of the activity, they were not fully prepared for an exposure to an interdisciplinary learning experience.

Other observers said that although they struggled to apply their mathematical knowledge within JS-Eden, the students were not afraid to use the environment to try out their answers (e.g. Group C's line drawing experiments in Figure 6). In a previous study (Beynon and Harfield, 2010), a high school student commented that you don't actually need much knowledge of the Eden environment to use it to find solutions to the tasks.

A week after the session, students were asked to reflect on their experience in the form of a student diary. The diary requested students to comment on "what went well", "what did not go well", "what did you like the most", and "what did you like the least (dislike)". Of the 25 students in the class, 20 student diaries were collected. Many of the positive comments were focussed on the environment, the group of experts and the English practice. The most negative comments were that there was too much content to cover in the time available, that the level of difficulty was too

high, that the environment contained bugs or user interface issues, and that using English was a barrier to communication.

Some of the students noticed the synergy with software development and expressed enthusiasm in building programs: "Finally we did some serious programming!" and "I was involved in serious programming tasks!". Many of these students had already been exposed to Scratch or Logo, and so their comments seem to corroborate the teacher's view that JS-Eden has some characteristics that make it feel more like "grown-up" programming than other educational environments.

Although the students said that the tasks were difficult, their diaries suggest that the tasks that the students enjoyed were some of the most difficult. One female student commented that "the animation" was what they liked the most, and another female student said "the planets moving around the sun was the one that I mostly liked".

The students recognised that the tasks were difficult, but they were mostly able to overcome obstacles, and the fact that they could do this was satisfying. "It was very difficult but we made it" and "We followed the worksheet, made notes on it and we managed to understand the programming language". Another student complained that one task "did not go well because it was very difficult, I struggled at finding a solution and I did not submit the task on time", but after that he said, "I liked it when we were making the figures. It was fun and enjoyable!". Is this a contradiction? No, we think that it is an excellent case of the 'hard fun' Papert proposed as the essence of good education (Caperton, 2005).

Discussion: JS-Eden vs procedural constructionist environments

While the positive results of the trial no doubt share common elements with studies of other popular constructionist environments, there are noticeable conceptual differences in their approaches to creating software. Most constructionist environments have a necessary programming component, either procedural or object-oriented, that must be mastered (e.g. Scratch). The three activities noted by Ackermann's are evident in these environments, as part of a two stage process of designing and executing. "Instructing" involves writing some instructions and then running; the "animating" and "modulating" activities are the same. JS-Eden offers a more direct approach as there is no two stage process to these activities. Interaction in JS-Eden is more like using a spreadsheet than writing a traditional program. When a learner "instructs" something to happen in JS-Eden, that is all they do because they are simply creating a connection. When a learner is "modulating", they are also tweaking a connection directly. This 'directness of interaction' is a characteristic feature of making construals with JS-Eden.

Within the short time of the workshop, the students undertook a number of tasks that involved applying existing mathematical knowledge to solve the problem of drawing and animating objects on the screen. One example was when they first struggled to apply Pythagoras's theorem in practice, but were able to realise and correct their mathematical misunderstandings, syntactic errors and tool misconceptions with the help of JS-Eden. Admittedly, their realisation was supported by a team of assistants who were there to provide hints and act as scaffolders in the learning process. However, the important point is that JS-Eden enabled students to tackle their domain learning within a relatively short period of time and with little prerequisite knowledge of programming.

Some of the students were able to complete the animation of the Earth revolving around the Sun in the final tasks. A Logo or Scratch equivalent of rotating the Earth around the Sun might involve first finding the starting position in relation to the Sun and then looping over a set of actions that include moving forward a small step and turning by a small amount (e.g. 1 degree). Furthermore, to animate more than one planet with the correct relative speeds would involve a significant number of extra calculations. In JS-Eden, the students were able to create these animations by making connections between the day of the year and the position of the planets in the construal. For example, if the Earth moves once around the sun in 365 days then after 182 days it will be

approximately half way (or 180 degrees), whereas Mars (with an orbit period of 687 days) has moved less than a quarter of its orbit. The connections made in JS-Eden have a more meaningful association with the domain than the individual procedural steps of moving an arbitrary small distance forward and turning. Being able to find and model a 'meaningful association' closely is one of the principles behind JS-Eden.

Concluding remarks

This small-scale study presents a setting where students were introduced to JS-Eden and 'making construals' for the first time. They were asked to use dependency to make connections between observables, to create simple construals that are representative of a situation or domain problem, and to reflect upon their experiences. While the students reported that the activities were challenging, the evidence suggests that they overcame their difficulties either in their group or with assistance. The interaction recordings demonstrated that the challenge may in part have been the friction between learning how to make construals and applying domain knowledge from mathematics. In some way, this can be taken as a positive in that there was a reasonably low barrier of entry to learning JS-Eden, and that domain learning is achievable after a short introduction to the tool. It provides some support for the claim that the making construals approach is accessible to young people for constructionist activities.

Furthermore, the making construals approach offers a more accessible environment for the kind of 'programming' activities of digital natives that were defined by Ackermann. Firstly, "instructing", "animating" and "modulating" are all activities that are involved in the direct manipulation of observables and dependencies in JS-Eden. Secondly, "instruction", "animating" and "modulating" are activities that might evoke a meaningful association between the software in the computer and a situation or object in the world. While there is typically a good correspondence between the products of computational thinking andprocedural tasks (e.g. performing a sequence of steps), the approach taken by JS-Eden of making connections between observables is well-suited to modelling a wide variety of everyday situations (cf. the movement of one object rotating around another). It is the principles of directness of interaction and meaningful association that we wish to emphasise in proposing making construals with JS-Eden as an attractive alternative to the technocentric constructionist tools that rely on computational thinking.

References

Ackermann, E. K. (2013) *Programming For The Natives: What is it? What's In It For The Kids?* Constructionism 2012, National & Kapodistrian University of Athens, Greece.

Beynon, M. et al. (2015) *Making construals as a new digital skill: dissolving the program - and the programmer - interface*. Proceedings of the Interactive Technology and Games (ITAG) conference, Nottingham Trent University, 22-23 October 2015.

Beynon, M. and Harfield, A. (2010) *Constructionism through Construal by Computer.* Constructionism 2010, The American University of Paris, August 2010

Beynon, M. (2012) *Modelling with experience: construal and construction for software.* Chapter 9 in Ways of Thinking, Ways of Seeing (ed. Chris Bissell and Chris Dillon), Automation, Collaboration, & E-Services Series 1, Springer-Verlag, January 2012, ISBN 978-3-642-25208-2, 197-228.

Brennan, K. (2015) *Beyond Technocentrism: Supporting Constructionism in the Classroom.* Constructivist Foundations 10(3): 289–296.

Caperton, I. (2005) *For Seymour Papert "hard fun" is the essence of good games AND good education.* Telemedium: the journal of media literacy. 52 (1 & 2) 16-19. Madison, WI: National Telemedia Council.

Harfield, A. and Beynon, M. (2012) *Empirical Modelling for constructionist learning in a Thai secondary school mathematics class.* In the 9th International Conference on eLearning for Knowledge-Based Society (eLearningAP 2012), Siam Technology College, Thailand, 13-14th December 2012.

Papert, S. (1980) *Mindstorms: Children, computers and powerful ideas.* Basic Books, New York, USA.

Utting, I., Cooper, S., Kölling, M., Maloney, J., and Resnick, M. (2010) *Alice, Greenfoot, and Scratch: A Discussion.* ACM Transactions on Computing Education, Vol. 10, No. 4, Article 17, November 2010.

Wing, J. M. (2006) *Computational thinking.* Communications of the ACM 49 (3): 33.