# FrameIT Reloaded: Serious Math Games from Modular Math Ontologies

Denis Rochau, Michael Kohlhase, and Dennis Müller

Computer Science, Jacobs University, Bremen, Germany

**Abstract.** Serious games are an attempt to leverage the inherent motivation in game-like scenarios for an educational application and to transpose the learning goals into real-world applications. Unfortunately, serious games are also very costly to develop and deploy. For very abstract domains like mathematics, already the representation of the knowledge involved becomes a problem.
We propose the **FrameIT** method that uses OMDoc/MMT theory graphs to represent and track the underlying knowledge in serious games. In this paper we report on an implementation and experiment that tests the method. We obtain a simple serious game by representing a "word problem" in OMDoc/MMT and connect the MMT API with a state-of-the-art game engine that "plays" the problem/knowledge exploration process.

## 1 Introduction

Serious games could be a potential solution to the often-diagnosed problem that traditional education via personal instruction and educational documents has serious scalability, subject specificity, and motivation limitations. A serious game is *"a mental contest, played with a computer in accordance with specific rules, that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives"* [Zyd05]. Serious games have the power to effectively supplement technical documents and online courses and thereby allow students to learn how to apply their knowledge to real world scenarios. Moreover, serious games very elegantly solve the motivation problem many people experience when studying technical subjects. Through gamification [Det+11] a serious game can be very entertaining while at the same time providing educational value to the user. Unfortunately, serious games are currently either very complex, domain specific, and expensive applications or lack the necessary complexity to effectively facilitate learning of complex subjects like mathematics.

To alleviate this, we propose the **FrameIT** method [KK12] which uses mathematical knowledge management techniques (MKM) to map a real world scenario to its theoretical basis. In this paper we report on an implementation and experiment that tests the method. We obtain a very simple serious game by representing a "word problem" in OMDoc/MMT and connect the MMT API with a state-of-the-art game engine. This paper is a short version of [Roc16], to which we refer for details.

## 2 Preliminaries

Before we can explain the **FramelT** method in detail we need to establish a clear and common understanding of the **OMDoc/MMT** format and **Learning Object Graphs** as they form the basis of the method.

## 3 Learning Object Graphs as OMDoc/MMT Theories

To realize our implementation we are going to use the OMDoc/MMT language [RK13], which is implemented **MMT** system (Meta-Meta-Tool) [Rab13], as a foundation independent logic framework that allows us to create logic powered applications. OMDoc/MMT consists of theories, symbols, and objects which are related to each other by typing and equality. A theory in OMDoc/MMT is defined as a list of symbol declarations, where each symbol declaration is of the form $c[: t][= d][\#N]$, where $c$ is the symbol identifier, the objects $t$ and $d$ are its type and definiens, and $N$ its notation. MMT objects over a theory are formed from the symbols available to the theory. Moreover, theories can represent, via the Curry-Howard correspondence, judgments, inference rules, axioms, and theorems. A theory morphism for two theories $A$ and $B$ is a morphism $m : A \to B$ that relates both theories to each other by mapping every symbol from the theory $A$ to a symbol in the theory $B$.

Theories and theory morphisms together form multigraphs, which we call **theory graphs**, that relate different theories to each other. In such a theory graph we have two different kinds of edges: **views** and **inclusions**. Inclusions allow us to combine and reuse multiple theories by including them in an inheritance style fashion while views allow us *"to link two pre-existing theories"* given that *"all the source axioms are theorems of the target theory"* [Koh14]. Furthermore, inclusions cannot form a cycle in a theory graph, while views can.

We will use theory graphs as **learning object graphs** in the FramelT context. They form the fundamental basis for the FramelT method as they allow us to relate different learning objects with each other in a machine understandable and logical way [KK12].

As theories and theory morphisms form a category with certain colimits, MMT is able to derive a pushout from two theories $A$ and $B$ it exists. A pushout takes two theory morphisms $f : C \to A$ and $g : C \to B$ and produces a theory $P$ and two morphisms $i_1 : A \to P$ and $i_2 : B \to P$ such that the square commutes (Figure 1). Intuitively, the pushout $P$ is formed as the union of $A$ and $B$ so that they share exactly $C$. [Rab15]

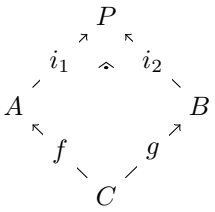



**Fig. 1.** Pushout

In the current version of MMT the result of a pushout is a new MMT theory that contains a set of simplified declarations. Lastly, MMT provides us with several ways of developing services and applications on top of it, we can either use the RESTful interface, develop MMT plugins in Java/Scala [Rab13] or directly execute a Scala script via the MMT shell.

### 3.1 Unreal Engine

The Unreal Engine is a game engine that provides game developers with the necessary tools to design and build games, simulations, and visualizations [Inc16]. The provided tools include a rich game editor that lets the developer to manage game assets, create the game user interface and build up the game environment.

The functionality of a game can be developed in C++ and/or by blueprint visual scripting. Another important feature of the Unreal Engine 4 is its platform independence, which allows us to deploy our project to nearly all platforms. This is crucial for future development as it enables us to target a larger audience. The engine supports plugins and the complete source code of the engine is available on GitHub, which could allow us in the future to include support for MMT natively.

## 4 The FramelT Method

The central idea of this new approach to serious games is to use the **FramelT** method to frame real world problems in our serious game environment as abstract problems [KK12]. We have extended and refined this method to address the shortcomings revealed during its implementation.

The primary objective of this method is to provide students with a new way of applying their domain knowledge to real problems. It is not intended as the primary way of obtaining new knowledge. Students are still expected to acquire new knowledge through personal instruction, by reading technical documents or by taking online courses. Nevertheless, our method helps the user to retain and understand the learned material. To exemplify the FramelT method we want to illustrate its application on a concrete example problem – finding the height of a tree (see Figure 2).

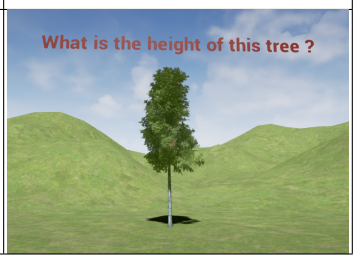
| Word Problem | Game Problem |
|---|---|
| How can you measure the height of a tree you cannot climb, when you only have a laser angle finder and a tape measure at hand? |   |

**Fig. 2.** Example Problem

In this example, we assume that the user has recently learned about trigonometry. Thus, the user might realize, after thinking about the problem, that they can map the real world problem to a simple trigonometry problem. This process of discovering a mapping between real world problems and abstract problems is exactly what the user should learn/train by playing our serious game.

This problem is similar to what many students might find in their mathematics textbook when they first learn about trigonometry. In addition, many textbooks might even provide small diagrams to visualize the problem. While this

method of presenting problems to students is acceptable for elementary problems, it is an inadequate approach for more complicated compound problems which are harder for students to understand and solve. Additionally, traditional mathematics textbook problems often focus on the computational aspects of solving problems, rather than on the more conceptual aspects that are underlying these computational solutions.

Our method allows students to explore a problem in its entirety in the game world. We will see below that being able to explore the game world and our method itself allows students to understand and solve even more complex problems in a step by step fashion. Furthermore, the focus of our method is not to train the ability of students to evaluate mathematical expressions, but instead their ability to apply their knowledge to real world problems. Thus, in our game any computational results are provided by MMT or the game itself.

Figure 3 shows the complete learning object graph[1] associated with the problem from Figure 2. It is partitioned into a game world side and MMT side. The game world side shows the different parts of the serious game the user is able to interact with, while the MMT side contains the learning object graph that powers the serious game and allows our approach to work. Additionally, the MMT side is subdivided into two separate sections, where the first section represents the current user knowledge and the second section represents the new knowledge that should be obtained by the learner. Actually, this separation is made for explanatory purposes only. In practice these sections are part of the same theory graph.

### 4.1 Game World Side (User Perspective)

On the game world side, the user is prompted to solve a given problem, here to find the height of a tree. Before trying to solve the problem at hand the user is able to explore his or her surroundings to accustom himself or herself with the problem and to think about possible approaches. In particular the user has to register facts about the world they are in. In the beginning, a user knows nothing about the world and so their list of registered facts is empty. However, the user can obtain new facts about the world by using scrolls or the a set of gadgets provided.

In our method scrolls are a mechanism to obtain new facts about the world from existing ones. A **scroll** is a game object that contains mathematical explanations and conditions that need to be satisfied in order for the given scroll to produce a new fact about the world. The name "scroll" is meant to evoke the fact that the knowledge contained in it is a valuable commodity in the game. Eventually scrolls are objects that can be found, traded, and earned. From a logical perspective a scroll acts like a complex inference rule that encapsulates a theorem or definition. To apply a given scroll, the user has to map a subset of the facts they obtained about the world to the contents of the specific scroll.

[1] We have glossed the contents of the theories and partially visualized them by diagrams, for the actual content see Figure 4.

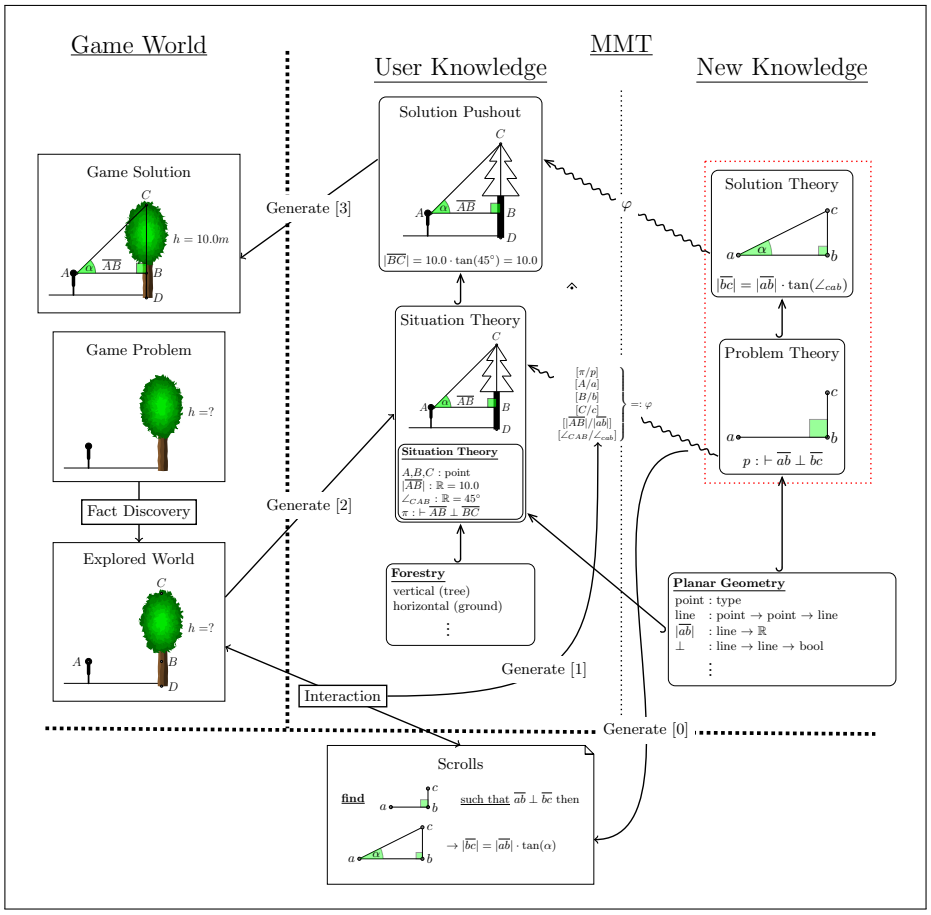



**Fig. 3.** Learning Object Graph Example

As scrolls require facts as their input, the user has to first use some of the provided gadgets to determine properties of the world and the objects within it. A **gadget** is a game object that can either produce new facts solely by interacting with the world itself or by relating existing facts to each other and thereby providing additional information. For instance, the **pointer gadget** marks a point in the game world and produces a new fact, which simply declares the existence of a point with the specified name. Based on this the user can relate two different point facts to each other by measuring the distance between them with a **measuring tape** gadget. The **laser angle finder** gadget to measure angles between three different marked points. The result of the exploration is visualized on the left hand side of Figure 3.

Didactically, the game world situation is rigged so that the user cannot solely rely on the provided gadgets to solve the given problem. Instead they need to

use scrolls to discover new facts about the world. In our example, the user cannot climb the tree and therefore not measure its height directly with the provided measuring tape gadget. In the problem at hand, the user could chose the trigonometry scroll at the bottom Figure 3 which provides the length of the opposite side of a right angle triangle given the angle and the length of the adjacent side.

In this situation, the FramelT methods works as follow: The user uses the marking gadget to mark multiple points in the game world, such as a point $A$ at his or her current location, a point $B$ at the base of the tree and a point $C$ at the top of the tree. The user proceeds by using the laser angle finder gadget to measure the angle between the points $C$, $A$ and $B$ and the measuring tape is used to determine the distance between the point $A$ and the $B$. Lastly, the user maps the obtained facts to the contents of the scroll by assigning points $A$, $B$, $C$ to the required point $a$, $b$, and $c$ in the scroll. If the perpendicularity condition is met, the scroll is applicable and will produce a new fact, otherwise it will provide the user with an explanation of why the scroll was not applicable[2] Finally

Lastly, the user has to assign a given fact as the solution of the problem. In our case they have to declare that the newly obtained fact about the distance between the points $B$ and $C$ represent the height of the tree. The game is then going to check this answer and if the user solved the problem correctly it is going to congratulate the user on the correct solution. Otherwise, the game will show them the flaws in their approach in order to facilitate learning.

### 4.2 OMDoc/MMT: Framing Situations and Pushing Out Solutions

The concrete knowledge registered by the user and abstract mathematical knowledge are jointly represented in a modular theory graph in OMDoc/MMT – see the right side of Figure 3.

We formalize the current list of registered facts into the **Situation Theory**, which is extended after any exploratory action of the user: Each discovered fact is translated from its representation in the game to a symbol declaration in MMT. For instance, the marked point $A$ in the game world has underlying data structures associated with it and through this generation step we distill the elaborate game data structures into one symbol declaration, that declares a new constant symbol $A$ of type *point*.

To fix the meaning of declarations in the situation theory, it includes several other theories by using **inclusion theory morphisms**. These imported theories are either theories that contain additional user knowledge about the world or base theories that provide us with the essential declarations to build up more sophisticated theories. In our example one of the included user knowledge theories could be a **forestry theory** which specifies properties of forests and trees, known to the user, such as that a tree is vertical and the ground is normally

[2] Generating helpful explanations in case of failure was left to future work. It is clear that scrolls can be extended suitably, e.g. via the techniques put forward in [GM08].

horizontal. A common base theory could be a **planar geometry** theory that provides us with declarations for points and lines.

Thus a situation theory and its included theories only represent the current user knowledge about the world. To discover new facts about the world we need to frame the current user knowledge to a **problem/solution pair** [KK12] that produces new facts about the world. A problem/solution pair consists of two theories, the **problem theory** and the **solution theory** and an inclusion morphism between them. A problem theory contains a formal definition of an abstract (mathematical) problem, while the solution theory contains the corresponding solution to the problem. In other words, the problem theory specifies the required facts that are needed as an input for a scroll and the solution theory specifies the new facts that will be obtained as an output. Therefore, a problem/solution pair acts similarly to a mathematical function. In Figure 3 one can see an example for a problem/solution pair in the red box on the right hand side. Similar to the situation theory, the problem theory is supported by the inclusion of several other theories such as the planar geometry theory. In fact these are important for the application of the mathematics to the situation.

In order to compute a new fact from a given problem/solution pair, we need to create an assignment ($\varphi$) that maps the given information from the situation theory to the correct problem theory. Moreover, this assignment can be understood as specifying the input parameters to a mathematical function. As described in subsection 4.1 the user specifies this assignment with his or her interaction between the chosen scroll and the explored world. On the MMT side, this interaction is used to generate a view between the situation theory and the problem theory. In our example, this view needs to assign a symbol declaration from the situation theory to the corresponding symbol declaration in the problem theory. For instance, the points ($A$, $B$, $C$) in the situation theory are assigned to the respective points ($a$, $b$, $c$) in the problem theory. Obviously, the view between a problem theory and a situation theory is going to depend purely on which scroll a user chooses, as the assignment will be generated from this interaction.

From the generated situation theory, the appropriate problem/solution pair and a view between the situation theory and the problem theory, MMT is able to produce a corresponding **solution pushout**. A pushout is only successfully produced if the view between the situation theory and the problem theory is total and appropriate. For instance, if we generate the same view but the lines $AB$ and $BC$ are not perpendicular in our situation theory, then the problem/solution pair is not applicable, because the assignment fails to be a total view. Fortunately, the solution pushout is successful in our example and the resulting theory contains a new symbol declaration that defines the length of the line segment $\overline{BC}$. Lastly, the solution pushout theory is used to add the newly obtained facts to the list of facts in the game itself. These newly obtained facts can then be used by the user to either solve the problem at hand or as the inputs for further scroll applications.

### 4.3 Scrolls

Scrolls allow the user to discover new facts about the world. Thus, they form the primary link between the Game World and the logic engine. Given the need for scrolls, we would like to generate them from the predefined problem/solution pairs.

While the current form of problem/solution pairs works perfectly for our present implementation, for future implementations and for the generation of scrolls they need to be extended and refined. The goal behind this extension would be to include visualization theories that allow us to display the abstract problem and its solution effectively in the game as well as on the scroll itself. Such an extension of problem/solution pairs could look similar to Figure 4.

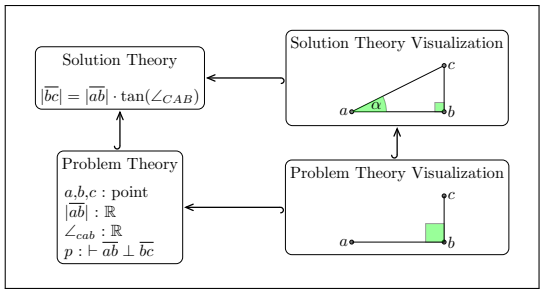



**Fig. 4.** Problem Solution Pair Extension

A scroll consists of two major parts. The first part of a scroll is a document that contains mathematical explanations in natural language in addition to diagrams that illustrate the concepts. This document could be generated non-dynamically by an extension of sTEX [Koh08] we tentatively call **ScrollTEX**. In the game itself this document should be displayed each time the user inspects the scroll. The second part of a scroll is a set of machine readable information that specifies exactly what kind of facts a scroll requires for its application and what kind of facts it is going to produce as an output. The OMDoc format [Koh06] might be the ideal format here as it allows us to store formal and informal information in a highly interrelated form and generate active mathematical documents [Koh+11].

The FramelT method supports compound problems directly. Each subproblem of a compound problem can be solved by discovering the required facts for the solution of the subproblems through successive scroll application and tool usage. The facts representing the solution of the subproblems can then be related to each other in exactly the same way, which in turn allows the user to solve the problem in its entirety. In this section we presented a simplified example, but it is not difficult to think of more advanced examples that require multiple scroll applications.

## 5 Design and Implementation

To evaluate the refined FramelT method, we designed and implemented a proof of concept serious math game that demonstrates our method. Moreover, this

first implementation allows us to critically assess the method and to discover any shortcomings.

### 5.1 MMT

On the MMT side we created the respective theories for the theory graph seen in Figure 3. The basis (meta-theory) of all of our theories is formed by a higher-order logic theory, which allows us to declare multiple different types and provides us with the basic logical quantifiers and semantics.

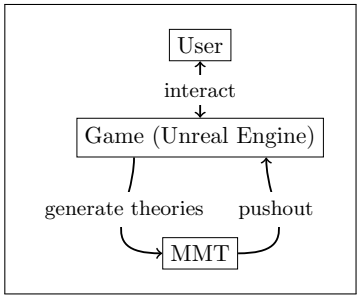



**Fig. 5.** System Architecture

**Fundamental Theories** For the FramelT method we require a set of fundamental theories on which we can base the higher level theories such as the situation theory or the problem/solution pairs. Thus, we implemented a real number theory and an Euclidean geometry theory for our proof of concept implementation.

The geometry theory makes use of the real number theory to implement other types such as vectors and lines in similar fashion. Given these two theories we are able to form precise propositions about the world and the problems within it. Obviously, these fundamental theories are incomplete and in order for them to handle more complicated scenarios they need to be extended in the future.

**Problem Solution Pairs** The problem theory simply (see Figure 3) contains a list of propositions that are needed by the solution theory to solve the problem. The solution theory includes the problem theory and contains additional declarations that specify the solution to the abstract problem. In other words the problem theory specifies the "input" of a function while the solution theory specifies the "output". The problem and solution theory implementations can be seen in Figure 6. Although our implementation is based on the example presented in section 4, we deviate slightly from it as we are actually requiring the user to proof explicitly that the scroll is applicable. Hence, our problem/solution pair contains an additional declaration (an axiom), the angle between the ground and the tree, which needs to be provided by the user.

**Situation Theories and Framing** In order to determine the requirements for the situation theory and the view between the problem theory and the situation theory we implemented an example situation theory and view. The situation theory contains the observed facts and their values and the example view assigns every declaration in the problem theory a declaration from the situation theory.

**namespace** http://cds.omdoc.org/FramelT M

**theory** problem_theory : ?geometry =
- Pa : Vec3D D
- Pb : Vec3D D
- Pc : Vec3D D
- anglePaPbPc_value : $\mathbb{R}$ D
- anglePaPbPc : $\vdash$ ($\angle$ Pa Pb Pc) $\doteq$ anglePaPbPc_value D
- anglePcPaPb_value : $\mathbb{R}$ D
- anglePcPaPb : $\vdash$ ($\angle$ Pc Pa Pb) $\doteq$ anglePcPaPb_value D
- lineSegPaPb_value : $\mathbb{R}$ D
- lineSegPaPb : $\vdash$ (lineSegmentLength Pa Pb) $\doteq$ lineSegPaPb_value D
- proof : $\vdash$ ((<Pa, Pb>) $\perp$ (<Pb, Pc>)) D M

**theory** solution_theory : ?geometry
- **include** = ?problem_theory D
- lineSegPbPc_value : $\mathbb{R}$ O = (tan anglePcPaPb_value) $*$ lineSegPaPb_value D
- lineSegPbPc : $\vdash$ (lineSegmentLength Pb Pc) $\doteq$ lineSegPbPc_value D M

**Fig. 6.** MMT Problem and Solutions Theories

### 5.2 Game (Unreal Engine)

We only focus on the parts of the implementation relevant to the FramelT method and not on the auxiliary parts such as the creation of the landscape.

The problem exploration gadgets are implemented classes in the grame engines based on as C$^{++}$ classes for the different kinds of facts a user can discover. Each of these classes contains methods that can serialize that fact into the OMDoc format. Moreover, the game stores each of these facts in a list which we call the **fact list**. Whenever the game needs to produce a situation theory it iterates through this list to serialize each fact to the OMDoc format.

The most important of those tools is the *point marking tool* as the player can use it to create a point fact by simply marking a location. This has two different marking modes. The first mode allows the user to simply mark any location he or she points to, while the second mode can be used to mark specific spots on what we call semantic actors. Figure 7 shows the *point marking tool* in action: Semantic Actors are game objects that contain additional information and methods that allow them to interact in a more specific way with the user and his or her available tools. To illustrate, the tree in our example problem is a semantic actor, which allows the user to mark exactly the bottom and top of the tree with his or her *point marking tool*. If the tree would not be a semantic actor then marking exactly the top and bottom of the tree would be incredible

difficult. Fortunately, it is a semantic actor which allows the user to select the points by simply marking a point close to the actual top or bottom of the tree.

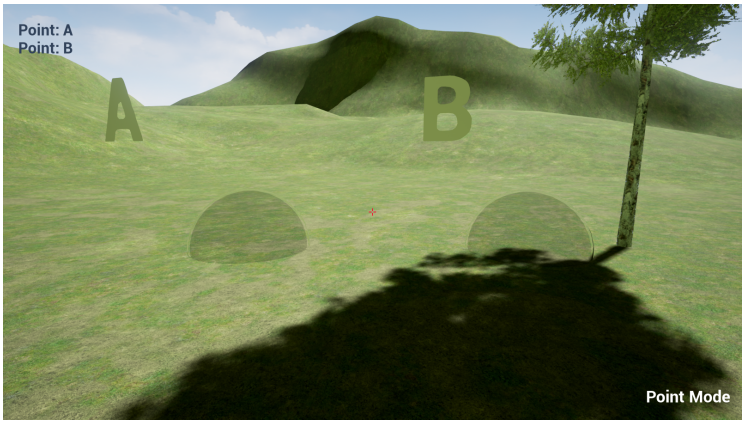



**Fig. 7.** Point Marking Tool

The measurements obtained with the *distance measuring tool* are in Unreal Units, which is the basic unit of length in the Unreal Engine. One Unreal Unit is equivalent to one centimeter. All of the facts the user discovers by using all of his or her tools are displayed as a list in the top left hand corner of the game.

As one can see in Figure 8 by measuring facts about the world the user's fact list grows gradually.

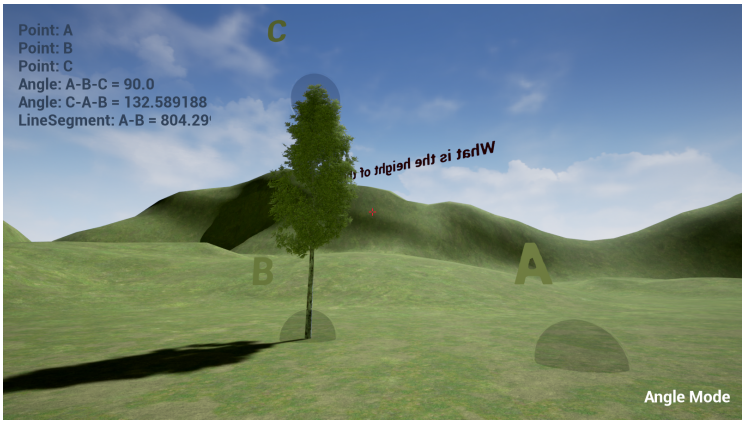



**Fig. 8.** Measuring Facts about the World

After measuring all facts the user can enter the *View Mode* in which they assign for each fact required by the scroll a fact from the fact list. During this assignment process a mapping between scroll facts and facts from the user's fact list is created. After the assignment is completed the mapping is serialized into OMDoc. At the same time the user's fact list is serialized into an OMDoc situation theory. Next, the generated OMDoc is passed on to MMT, which takes these theories and uses them to compute a solution pushout. If the pushout was successful then MMT produces a new theory that is subsequently parsed by the game. Through parsing this theory the necessary information needed to create the resulting facts is extracted and used to populate the fact list with the newly obtained facts. In case the pushout was unsuccessful an error message is displayed. The result of a successful pushout can be seen in Figure 9. In this specific case the user has now determined that the height of this tree is 875 centimeters and thus the user has successfully solved the problem.

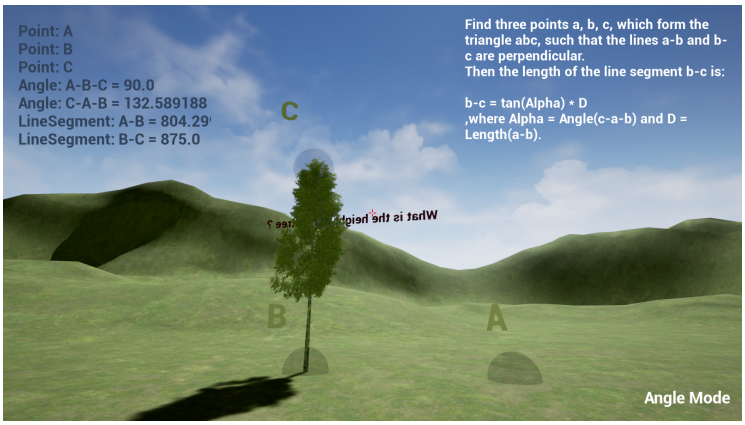



**Fig. 9.** Scroll Result

## 6 Conclusion and Future Work

The proof of concept implementation and our experiment the method shows that it is indeed possible to use mathematical knowledge management techniques (MKM) to map a real world scenario to its theoretical basis. This suggests that the FramelT method can be used as a framework for creating full-fledged serious math games.

*Separation of Necessary Skills* Crucially, the framework keeps many parts of a serious game generic. As a consequence, the development of new games (and extensions to existing ones) naturally separates into three independent tasks: the development of *i*) didactic game situations, *ii*) learning objects (real-world

problems, problem-solutions pairs and base theories) as mathematical theories in OMDoc/MMT, and finally *iii*) semantic actors and exploration gadgets. The first is a high-level, transdisciplinary design task that results in the storyboard for a serious game. For the implementation *ii*) only requires the skills of math educators trained in OMDoc/MMT formalization, while *iii* only needs regular game realization skills. In particular, the mathematics didactics and games realisation tasks are well-separated by the FrameIT framework, which promises to greatly simplify the realization phase. Based on our experience, realizing another "word problem-type" game problem is the work of one or two days for an individual trained in OMDoc/MMT formalization and we estimate the development of a new, non-trivial semantic actor or gadget to be in the same range. Of course this greatly depends on the availability of foundational theories and game components; fortunately, these are developments that can be shared and accumulated by an open community.

*Limitations* While we can already demonstrate the FrameIT method, we are still missing some integral parts such as the automatic generation of scrolls as discussed in section 4.3. Furthermore, we are currently not generating detailed explanatory texts in case MMT is unable to produce a pushout or the user was unsuccessful in solving the problem. Future implementations should address this by providing the user with helpful hints and error messages that allow the user to learn from his or her mistakes. Lastly, we would like to allow the user to be able to use his or her knowledge directly. For instance, a user should be able to create his or her own scrolls in the game by providing a proof that his or her envisioned scroll is mathematically sound.

Finally, the current lack of content does not yet allow us to demonstrate the ability of the FrameIT method to support compound problems. But the fact that scroll-based fact extension is essentially the same as theorem proving – which is one of the activities the MMT tool was designed for suggests that this should be very easy to achieve.

*Availability & Future work* Future work should also focus on the gamification aspects and the user interface of the game itself, as the current interaction between the user and the game is not ideal. For instance, the interface for assigning views is not optimal as we are selecting facts from a list of facts instead of selecting them directly in the game world. Hence, we would like to improve the user interface to make the game more intuitive and attractive for the user. Furthermore, we are currently not visualizing facts about angles or distances in the game world. Utilizing the results of human computer interaction research could allow us to resolve these issues effectively and thereby increase the usability of our serious math game.

While our implementation successfully connects MMT and the Unreal Engine, there still exists a plethora of software engineering challenges to make transferring knowledge between the system smoother and more efficient. Integrating MMT directly into the Unreal Engine could solve many of these challenges and

improve the performance of the system. Additionally, it would allow other serious game developers to use the services provided by MMT directly. Such an integration could use the Semantic Alliance framework [Dav+12] as its basis and adapt it for the Unreal Engine.

The current implementation is released under the MIT License and is available at https://github.com/KWARC/FrameIT, while the OMDoc/MMT content can be found at https://gl.mathhub.info/groups/FrameIT. As a first step we want to stabilize the system and extend the content (game situations and OMDoc/MMT formalizations) to a point, where we can start designing coherent serious games from the collection of problems.

*No Evaluation Yet* The work reported on in this paper only constitutes of proof-of-concept for the FrameIT method. In particular, the system is not currently in a shape that we could qualitatively and quantitatively analyze the usefulness and usability of the developed serious math game, which was the motivation of the FrameIT method in the first place.

*Acknowledgements* The development of the FrameIT method has profited from discussions in the KWARC group at Jacobs University, in particular from contributions by Mihnea Iancu and Andrea Kohlhase.

## References

[Dav+12] Catalin David et al. "Semantic Alliance: A Framework for Semantic Allies". In: *Intelligent Computer Mathematics*. Ed. by Johan Jeuring et al. LNAI 7362. Berlin and Heidelberg: Springer Verlag, 2012, pp. 49–64. URL: http://kwarc.info/kohlhase/papers/mkm12-SAlly.pdf.

[Det+11] Sebastian Deterding et al. "From Game Design Elements to Gamefulness: Defining "Gamification"". In: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*. MindTrek '11. Tampere, Finland: ACM, 2011, pp. 9–15. DOI: 10.1145/2181037.2181040.

[GM08] George Goguadze and Erica Melis. "Feedback in ActiveMath Exercises". In: *International Conference on Mathematics Education (ICME)*. 2008.

[Inc16] Epic Games Inc. *Unreal Engine 4*. https://www.unrealengine.com/. 2016. URL: https://www.unrealengine.com/ (visited on 02/27/2016).

[KK12] Andrea Kohlhase and Michael Kohlhase. "Frames: Active Examples for Technical Documents". 2012. URL: http://kwarc.info/kohlhase/submit/activeex-2012.pdf.

[Koh+11] Michael Kohlhase et al. "The Planetary System: Web 3.0 & Active Documents for STEM". In: *Procedia Computer Science* 4 (2011): *Special issue: Proceedings of the International Conference on Computational Science (ICCS)*. Ed. by Mitsuhisa Sato et al. Finalist at the Executable Paper Grand Challenge, pp. 598–607. DOI: 10.1016/j.procs.2011.04.063.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. LNAI 4180. Springer Verlag, Aug. 2006. URL: http://omdoc.org/pubs/omdoc1.2.pdf.

[Koh08] Michael Kohlhase. "Using LaTeX as a Semantic Markup Format". In: *Mathematics in Computer Science* 2.2 (2008), pp. 279–304. URL: https://svn.kwarc.info/repos/stex/doc/mcs08/stex.pdf.

[Koh14] Michael Kohlhase. "Mathematical Knowledge Management: Transcending the One-Brain-Barrier with Theory Graphs". In: *EMS Newsletter* (June 2014), pp. 22–27. URL: http://www.ems-ph.org/journals/newsletter/pdf/2014-06-92.pdf.

[Rab13] Florian Rabe. "The MMT API: A Generic MKM System". In: *CoRR* abs/1306.3199 (2013). URL: http://arxiv.org/abs/1306.3199.

[Rab15] Florian Rabe. "Theory Expressions (a Survey)". submitted. 2015.

[RK13] Florian Rabe and Michael Kohlhase. "A Scalable Module System". In: *Information & Computation* 0.230 (2013), pp. 1–54. URL: http://kwarc.info/frabe/Research/mmt.pdf.

[Roc16] Denis Rochau. "FrameIT Reloaded: Serious Math Games from Logic". B. Sc. Thesis. Jacobs University Bremen, 2016.

[Zyd05] M. Zyda. "From visual simulation to virtual reality to games". In: *Computer* 38.9 (2005), pp. 25–32. DOI: 10.1109/MC.2005.297.