JACOBS
UNIVERSITY

FrameIT Reloaded:
Serious Math Games from Logic

by

Denis Rochau

Bachelor Thesis in Computer Science

Prof. Dr. Michael Kohlhase

Date of Submission: May 8, 2016

Jacobs University - School of Engineering and Science

With my signature, I certify that this thesis has been written by me using only the indicated resources and materials. Where I have presented data and results, the data and results are complete, genuine, and have been obtained by me unless otherwise acknowledged; where my results derive from computer programs, these computer programs have been written by me unless otherwise acknowledged. I further confirm that this thesis has not been submitted, either in part or as a whole, for any other academic degree at this or another institution.

Bremen, May 8, 2016

Abstract

Obtaining new skills and knowledge by learning is part of every human life. Unfortunately, there are many challenges to provide students with a good education such as the fact that personal instruction is not scalable and very expensive. Another challenge is that alternatives to personal instruction, like learning by reading technical documents, are often faced with the problem that they do not teach students how to apply their freshly obtained knowledge to real world problems.

To address these challenges and develop a solution to the presented problem I developed a serious math game using the **FramelT** method. This method was refined and extended during this research project and it allows the user of our serious game to explore and solve real world problems in an interactive way, thereby supplementing technical documents and online courses. While the developed implementation still has many limitations, it demonstrates clearly that the method itself is able to alleviate the presented problem. Therefore, this project lays the groundwork for a new generation of serious games in education which are based on logic and the **FramelT** method.

Contents

The road to wisdom? - Well, it's plain
and simple to express:

Err
and err
and err again
but less
and less
and less.

Piet Hein

Acknowledgements

I would like to thank Prof. Dr. Michael Kohlhase for his valuable guidance and inspiring supervision. Furthermore, I would like to thank Dennis Müller for aiding me in my research project as well as for the numerous discussions that helped me understand MMT.

1 Introduction

Learning is an integral part of every human life. We start exploring the world as children, before we, in most cases, continue our education by being taught many different aspects of our world by teachers and professors in school and university. However, people are acquiring new knowledge not only by listening to lectures and experiencing personal instruction, but also by a variety of different methods such as studying books, taking online courses, and playing serious games.

Obtaining knowledge by personal instruction in school and university is still the leading way to obtain an education, but the approach itself faces several problems such as the fact that personal instruction is very expensive and hard to scale. On the other hand, books and other technical documents as well as online courses provide a less expensive and more scalable way of obtaining an education. Unfortunately, they often suffer from the fact that the taught knowledge is presented in a static way that makes it difficult to use for solving problems in real world scenarios. To alleviate this problem, technical documents and online courses often contain examples which demonstrate how the knowledge can be applied to real world problems. Sadly, the provided examples are often unable to fully solve the problem as they are static and do not allow the user to explore the problem scenario in its entirety [KK12].

Serious games for education could be a potential solution to this problem. A serious game, as defined by Zyda [Zyd05], 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"*. 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 learning. Through gamification [DDKN11] 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 solve the presented problem and to provide students with a new way to apply their obtained knowledge to real world problems, I created the first complete implementation of the **FrameIT** method during this research project. This method was initially devised in [KK12] and a first very elemental implementation was developed by Daniel Rachev as part of his Bachelor Thesis [Rac12]. Moreover, the FrameIT method itself is a result of the proposed SimSaLaBim project [KK11], whose goal was to create serious games for STEM. This method allows us to use graph-structured representations of knowledge to map a real world scenario to its theoretical basis. Although the FrameIT method is domain independent, in this research project I focused on creating an implementation for the mathematics domain, as it especially suffers from the real world applicability problem. Additionally, it is important to note that the FrameIT method has evolved significantly since its first presentation in [KK12] into a more elaborate method.

2 Related Work

Except from the before mentioned SimSaLaBim project and Daniel Rachev Bachelor Thesis there is not a lot of related work to be found. Many educational projects and websites focus on providing practice exercises, which allow students to apply specific concepts to abstract problems. For example, the popular online lecture website Khan Academy features many practice exercises that allow students to practice their newly obtained skills [Aca16].

Other projects develop complete serious math games, but their focus often lies on teaching students elementary mathematics such as elementary arithmetic. For instance, the serious math game Mathbreakers focuses primarily on teaching young children the basic concepts of addition, subtraction, multiplication and division of integer numbers and fractions [Co.16]. While helping students master these elementary skills is very important, this research project focuses on teaching students how to solve real world problems.

One of the most directly related research projects is the ActiveMath project [MGLU09]. ActiveMath is a web-based learning environment that uses an intelligent tutoring system to teach students mathematics. The goal of the ActiveMath platform is to provide a ”truly interactive, exploratory learning” environment to the user and thereby facilitate learning [MS04]. A central component of the ActiveMath system is its course generator, which utilizes information about learning objects and the learner to generate a sequence of learning objects [MGLU09]. Similar to our research project, the ActiveMath project uses the OMDoc standard for mathematical documents to represent and store semantic knowledge.

3 Preliminaries

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

3.1 MMT

To realize our implementation we are going to heavily depend on **MMT**(meta-meta-theory/tool) as it is a foundation independent logic tool that allows us to create logic powered applications. Foundation independence is *”the systematic abstraction from logic-specific conceptualizations, theorems, and algorithms in order to obtain results that apply to any logic”* [Rab15]. This foundation independence makes MMT well suited for our system as its modularity allows us to focus on developing the envisioned application without having to worry about the design and implementation of the logic or tool itself.

The MMT language, which is implemented by the MMT API [Rab13], consists of theories, sym-

bols, and objects which are related to each other by typing and equality [Rab15]. A theory in 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 [Rab13]. 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 [Rab15].

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 are allowed to form cycles.

Theory graphs, called **learning object graphs** in this context, form the fundamental basis for the FrameIT method as they allow us to relate different learning objects with each other in a machine understandable and logical way [KK12]. In our approach a learning object is a representation of knowledge on a specific topic, which should be obtained by each student trying to learn about that specific topic. We will represent these learning objects as MMT theories, which are connected to other learning objects by theory morphisms. Inclusions are used to build up a modular representation of knowledge in a learning object graph similar to how inheritance is used in an object oriented programming language to build up a modular set of classes [KK12]. On the other hand, views are used *"to view a learning problem as one that is already understood"* [KK12]. This practice gives our method the name FrameIT as we are creating a theory morphism from the framing theory which represents our knowledge into the framed theory which represents our problem.

As theories and theory morphisms form a category, MMT is able to derive a pushout from two theories A and B if such a pushout 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. Intuitively, the pushout P is formed as the union of A and B so that they share exactly C. [Rab] This concept is visualized in Figure 1.

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 provided RESTful interface, develop MMT plugins in Java/Scala [Rab13] or directly execute a Scala script via the MMT shell.

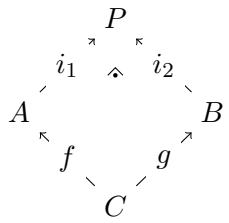

Figure 1: Pushout

3.2 OMDoc

The OMDoc (Open Mathematical Documents) format is an XML-based content markup scheme for mathematical documents [Koh06]. It can be used to represent formal as well as informal content, which allows the user to combine natural language descriptions with formal content such as formulae.

The OMDoc format uses three levels of modeling to represent mathematical knowledge. The highest level of representation is the *Theory Level*, which allows the clustering of mathematical knowledge into theories that can be related to each other by morphisms. The second level of modeling is the *Statement Level*, which provides us with the ability to declare axioms, definitions, theorems and other formal aspects of mathematical knowledge as well as informal aspects such as natural language descriptions of definitions. The lowest level of representation is the *Object Level*, which provides the necessary markup to structure mathematical formulae [Koh06].

Moreover, MMT can be used to generate OMDoc documents from theories declared in the MMT language. This feature is very useful as the OMDoc format provides the necessary structure to be easily read and processed by a computer. This is why we will use this format as the communication medium between our game and MMT.

3.3 Unreal Engine

The Unreal Engine is a game engine developed by Epic games that provides game developers with the necessary tools to design and build games, simulations, and visualizations [EG16]. The provided tools include a rich game editor that allows the developer to manage game assets, create the game user interface and build up the game environment. Figure 2 shows a screenshot of the game editor.

The functionality of a game can be developed in C++ and/or by blueprint visual scripting. Both methods have their advantages and disadvantages and combining both is often the most effective way. In general C++ is better suited for implementing the complex core data structures and algorithms as the performance is superior compared to visual scripting and programming complex behaviors is significantly easier. However, blueprint visual scripting is ideal for simpler parts of the game where performance is less of an issue such as the user interface of the game. Our game is implemented in the same manner as the complex data structures and algorithms are implemented in C++, while the user interface is powered by blueprints.

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. Moreover, the Unreal Engine ships with excellent documentation and tutorials that allow new developers to quickly get up to speed. Lastly, 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. The listed features make the Unreal Engine an ideal engine for this research project. This is why the current implementation of the FrameIT method was developed with

the latest version of the Unreal Engine (4.11.2).

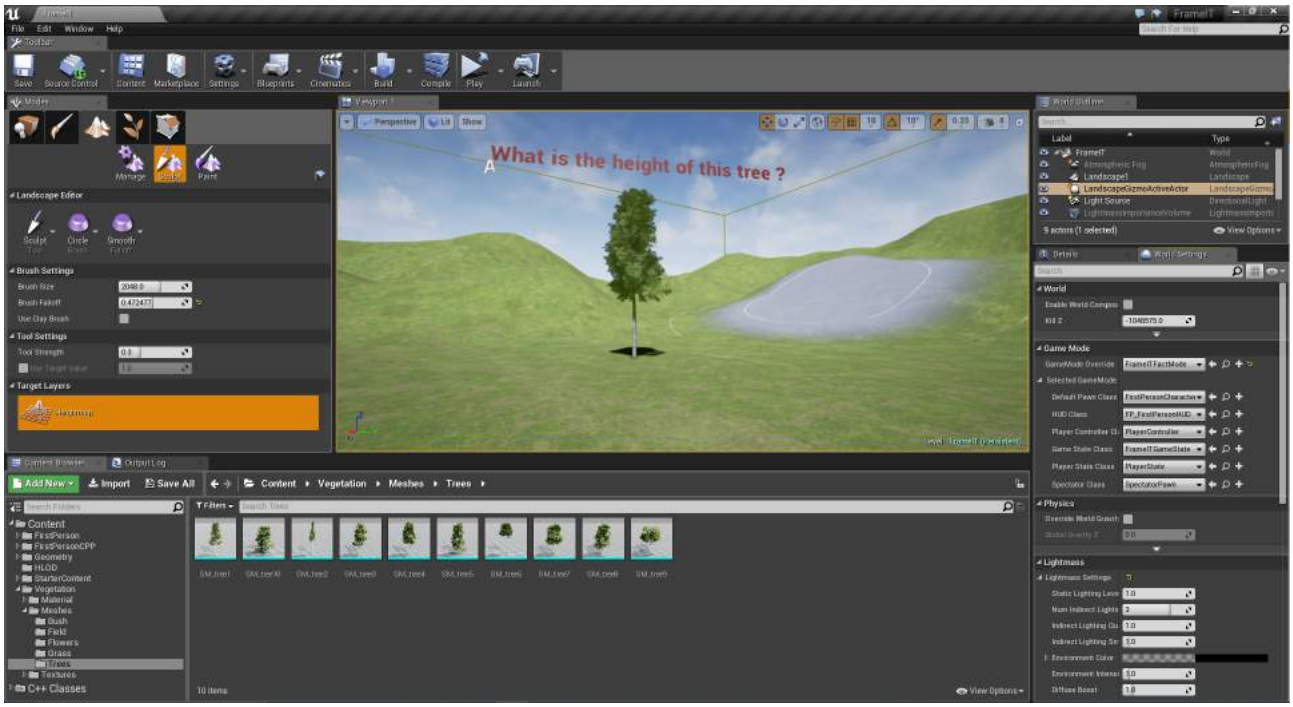

Figure 2: Unreal Engine Game Editor

4 Method: FrameIT

The central idea of this new approach to serious games is to use the **FrameIT** method to frame real world problems in our serious game environment as abstract problems. [KK12]. I 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. Therefore, 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 FrameIT method we want to illustrate its application on a concrete example problem. To simplify the example we are only interested in the height of the tree from the eye-height of the character in the game environment to the top of the tree.

In this example, we assume that the user has recently learned about trigonometry. Thus, the user might realize, after thinking about the problem, that he or she 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 by playing our serious 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?

Problem Representation in our Game:

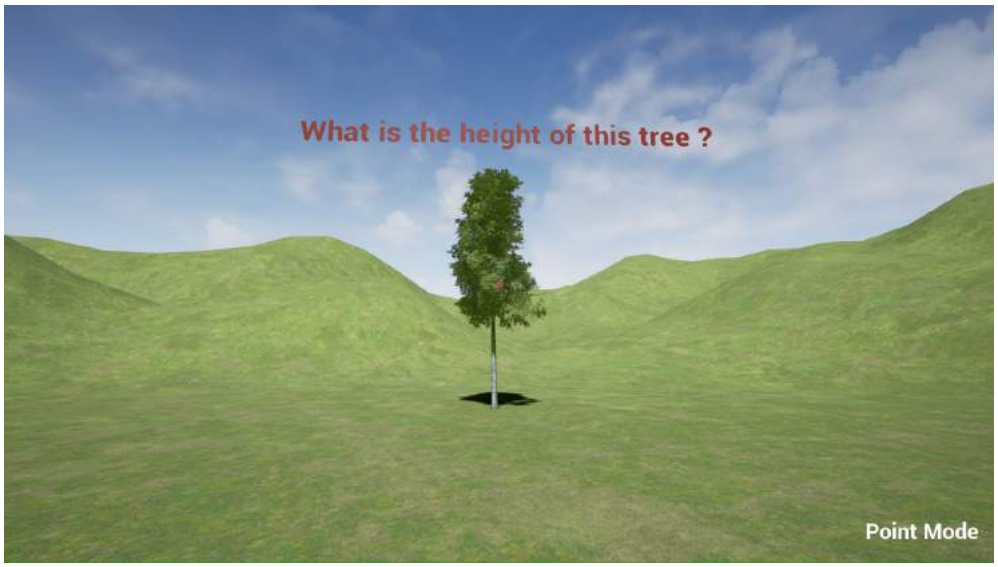

Figure 3: Example Problem

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, similar to the ones in Figure 4, to visualize the problem. While this means 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.

In contrast our method allows students to explore a problem in its entirety by exploring it in the game world. As we will see in section 4.4, 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 4 shows the complete learning object graph diagram associated with this specific problem. 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 actually 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.

Note, the theories on the MMT side in Figure 4 are not formally defined MMT theories, but rather serve as either example MMT theories or visual illustrations of the underlying formal theories.

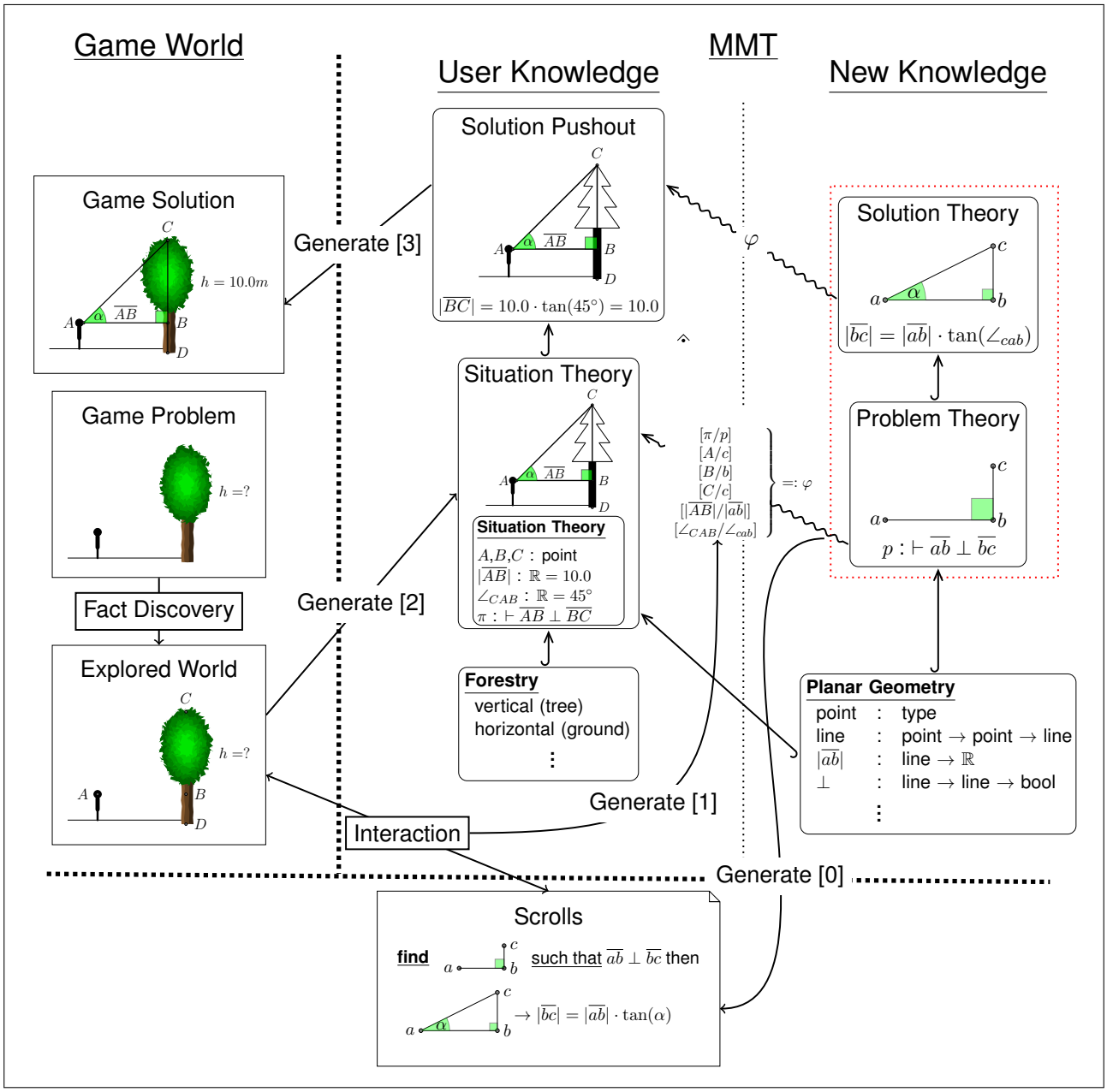

Figure 4: Learning Object Graph Example

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. To solve a given problem the user has to obtain facts about the world he or she is in. In the beginning, he or she knows nothing about the world and so his or her list of facts is empty. However, the user can obtain new facts about the world by using scrolls or the provided tools.

In our method scrolls are a mechanism to obtain new facts about the world from existing ones. Each scroll contains mathematical explanations and conditions that need to be satisfied in order for the given scroll to produce a new fact about the world. To use a given scroll, the user has to map a subset of the facts he or she obtained about the world to the contents of the specific scroll. In the game the user is provided with a collection of scrolls and each scroll contains a different approach.

As scrolls require facts as their input, the user has to first use some of the provided tools to determine properties of the world and the objects within it. These tools 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, by marking a point in the world the user produces a new fact, which simply declares the existence of a point with the specified name. He or she is then able to relate two different point facts to each other by measuring the distance between them with a measuring tape. Obviously, a measuring tape can only be used to measure the distance between points the user can physically reach, which means that the user is not able to directly measure the height of the tree as he or she cannot climb it. Another tool to relate different facts to each other could be a laser angle finder to measure angles between different marked points. This fact discovery through tools allows the user to reach a state of an explored world, where he or she observed and produced useful facts about the world. These facts can then be used as inputs for scrolls or to directly solve the given problem. This is visualized on the left hand side of Figure 4.

Didactically, the game world situation is rigged so that the user cannot solely rely on the provided tools to solve a given problem. Instead he or she needs 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. Nevertheless, the user can choose a scroll that contains an approach which allows him or her to compute the height of the tree from other facts. For instance, the user could chose the trigonometry approach provided by the scroll in the lower part of Figure 4. This approach provides the length of the opposite side of a right angle triangle given the angle and the length of the adjacent side. In addition, this approach requires that the tree in the game environment is vertical and that the ground is horizontal which make them perpendicular to each other.

In our example problem, the user realizes, after exploring his or her surroundings, that he or she wants to use the trigonometry scroll to determine the height of the tree. To apply the scroll the user has to obtain several facts about the world. Consequently, he or she uses the marking tool to mark

multiple points in the game world, such as a point (A) at his or her current location, a point (B) at eye-height on the trunk of the tree and a point (C) at the top of the tree. Each of these points represents a new fact that the user discovered about the world. While the existence of these points in itself is not directly helpful, relating them to each other by using the other tools, allows the user to gain new and more valuable facts about the world. After marking multiple points, the user proceeds by using the laser angle finder to measure the angle between the points C, A and B. Afterwards, the measuring tape is used to measure the distance between the point A and the B. Lastly, the user maps the obtained facts to the contents of the scroll by assigning a fact like the existence of a point A to the required point a in the scroll. After successfully assigning all required facts the scroll will produce a new fact if it was applicable. If it was not applicable, i.e. the required conditions where not met, it will fail to produce a new fact and instead provide the user with an explanation of why the scroll was not applicable. Generating helpful explanations in case of failure was not part of this research but is a feature for future work. Fortunately, in this example the required conditions are met, which allows the user to successfully obtain a new fact about the world.

Lastly, the user has to assign a given fact as the solution to the problem. In our cases he or she has 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 his or her correct solution. Otherwise, the game will show him or her the flaws in his or her solution in order to facilitate learning.

4.2 MMT Side

After explaining how the method would work from the user point of view, we will now continue by explaining the MMT side. The MMT side is composed of a theory graph, which is subdivided into a ”user knowledge part” and a ”new knowledge part”. The former contains the theories that represent the current knowledge of the user and the latter consists of the theories that contain the new knowledge that should be obtained by the user through playing the serious math game. As mentioned before this subdivision is made for explanatory purposes only.

In all situations we need to formalize our current list of facts into a theory, which we will call the **Situation Theory** (Generate[2]). This theory contains each so far discovered fact in formalized form and is regenerated 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*.

As the discovered facts form an dependency graph, the generation mechanism needs to ensure that all symbol declarations are produced in topological order. This means that symbol declarations that depend on other declarations need to be declared after their dependencies. To illustrate, the fact that there exists a point A and a point B needs to be declared before the dependent fact that the

distance between those points has a certain value. It is crucial to keep the topological order here as otherwise the resulting MMT theory would be ill-formed and thereby unusable.

Additionally, each situation theory is going to include 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 horizontal. A common base theory could be a **planar geometry** theory that provides us with declarations for points and lines.

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 a 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 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 4 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 order to compute a new fact from a given problem/solution pair, we need to create an assignment (φ) 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 going to be 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 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 (Generate[3]). 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 side and the MMT side. Given the need for scrolls, we would like to generate them from the predefined problem/solution pairs (Generate[0]). However, the generation of scrolls was not part of this research project, nevertheless I identified the necessary components of a scroll.

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 5.

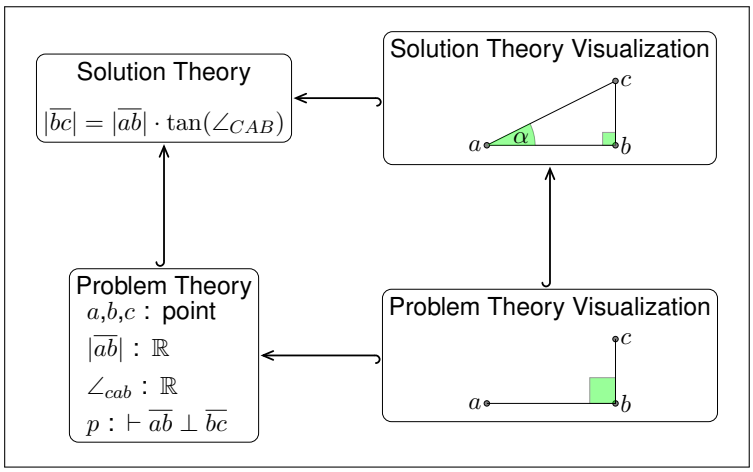

Figure 5: Problem Solution Pair Extension

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 should 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.

4.4 FrameIT as a Method for Learning

The FrameIT method facilitates learning by prompting the user to solve real world problems, which he or she can only solve by using scrolls. In the process of solving these problems the user starts to understand the theories and approaches presented in each scroll, which then slowly but surely over time causes the contents of a scroll to become part of the user knowledge.

Furthermore, the FrameIT 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 the provided example itself is actually a combination of several problems as the user has to find out the height h, the height from the ground up to his or her eyes and to reason that the absolute tree height is the sum of both.

5 Design and Implementation

Building up on the refined FrameIT method, I 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.

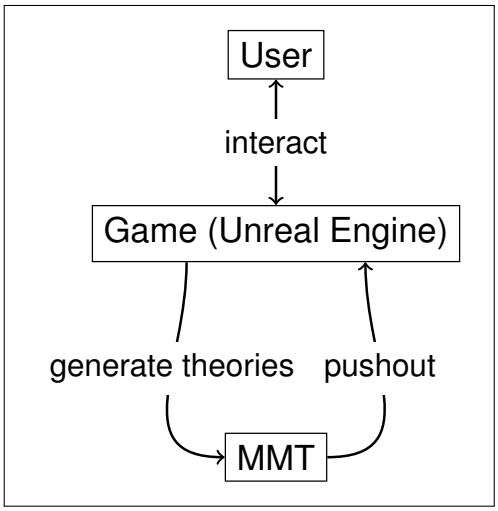

Figure 6: System Architecture

Equally to the FrameIT method the developed serious math game system consists of two main parts as shown in Figure 6. The user is going to play the game as described in section 4.1 and when he or she applies scrolls to obtain new facts about the world, the game is going to generate a corresponding situation theory. While we could theoretically provide MMT with additional information during this generation step, our generated situation theory contains exactly the facts the user discovered so far. From this generated theory as well as with the provided problem/solution pairs MMT will attempt to produce a pushout and send the result back to the game. Our proof of concept implementation poses a slightly different problem to the user than the one described in section 4. Instead of being interested in the height of the tree from the eye-height of the character to the top of the tree, we are interested in the real height of the tree. To support this, we provide the user with the ability to measure angles between arbitrary points.

5.1 MMT

On the MMT side I created the respective theories for the theory graph seen in Figure 4. The basis 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.

5.1.1 Fundamental Theories

For the FrameIT 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, I implemented a real number theory and a geometry theory for our proof of concept implementation. It is important to note, that I implemented a geometry theory instead of a planar geometry theory, due to the fact that the developed implementation keeps the problem in 3D and does not map it to its 2D variant. Therefore our implementation slightly deviates from the theory graph described in section 4.2.

In Figure 8 one can see an excerpt from the real number theory implemented in MMT. Essentially, we are declaring a new type of real numbers and defining functions such as addition and subtraction for this type. However, this only declares the existence of a new type and its associated functions. Furthermore, MMT is unable to recognize real number literals such a 42.0 directly. In order to utilize the real number type effectively we need to provide an additional set of semantic rules.

Fortunately, MMT allows us to define arbitrary literals as well as rules by injecting Scala code. These implemented rules are then executed when the respective expression needs to be evaluated. Figure 7 shows a code sample containing the implementation of the addition rule.

The geometry theory takes 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.

```
object RealLiterals extends RealizedType(Apply(OMS(tm),OMS(path ?
    "reals")),StandardDouble)

object Addition extends RealizedOperator(path ? "add") {
   val argTypes = List(RealLiterals, RealLiterals)
   val retType = RealLiterals

   def apply(args: List[Term]): OMLIT = args match {
      case List(RealLiterals(u : Double), RealLiterals(v : Double)) =>
          RealLiterals.apply(u + v)
      case _ => throw new Exception("Could not add the real numbers.")
   }
}
```

Figure 7: Scala Code for Real Number Theory rules

```
namespace http://cds.omdoc.org/FrameIT M

theory real_numbers : ?HOL =
    reals : tp D
    ℝ : type O
      = tm reals D
    negative : ℝ → ℝ O
             # - 1 prec 20 D
    add : ℝ → ℝ → ℝ O
        # 1 + 2 prec 5 D
    sub : ℝ → ℝ → ℝ O
        = [a, b] a + (-b) O
        # 1 - 2 prec 5 D
    ⋮
    rule info.kwarc.mmt.frameit.RealLiterals D
    rule info.kwarc.mmt.frameit.Negative D
    rule info.kwarc.mmt.frameit.Addition D
    ⋮
M
```

Figure 8: MMT Real Number Theory

5.1.2 Problem Solution Pairs

Based on the fundamental theories, presented in the previous section, I implemented the first problem/solution pair. The problem theory simply 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 9 and in Figure 10 respectively. 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 fact, the angle between the ground and the tree, which needs to be provided by the user.

```
namespace http://cds.omdoc.org/FrameIT M

theory problem_theory : ?geometry =
    Pa : Vec3D D
    Pb : Vec3D D
    Pc : Vec3D D

    anglePaPbPc_value : ℝ D
    anglePaPbPc : ⊢ (∠ Pa Pb Pc) ≐ anglePaPbPc_value D

    anglePcPaPb_value : ℝ D
    anglePcPaPb : ⊢ (∠ Pc Pa Pb) ≐ anglePcPaPb_value D

    lineSegPaPb_value : ℝ D
    lineSegPaPb : ⊢ (lineSegmentLength Pa Pb) ≐ lineSegPaPb_value D

    proof : ⊢ ((<Pa, Pb>)  (<Pb, Pc>)) D
M
```

Figure 9: MMT Problem Theory

```
namespace http://cds.omdoc.org/FrameIT M

theory solution_theory : ?geometry =
    include ?problem_theory D

    lineSegPbPc_value : ℝ O = (tan anglePcPaPb_value) * lineSegPaPb_value D
    lineSegPbPc : ⊢ (lineSegmentLength Pb Pc) ≐ lineSegPbPc_value D
M
```

Figure 10: MMT Solution Theory

5.1.3 Situation Theories and Framing

In order to determine the requirements for the situation theory and the view between problem theory and the situation theory I 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. Figure 12 and Figure 11 show the respective implementation of these theories.

```
namespace http://cds.omdoc.org/FrameIT M

theory situation_theory : ?geometry =
    pA : Vec3D D
    pB : Vec3D D
    pC : Vec3D D

    anglepApBpC_value = 90.0 D
    anglepApBpC : ⊢ (∠ pA pB pC) ≐ anglepApBpC_value D

    anglepCpApB_value = 45.0 D
    anglepCpApB : ⊢ (∠ pC pA pB) ≐ anglepCpApB_value D

    lineSegpApB_value = 100 D
    lineSegpApB : ⊢ (lineSegmentLength pA pB) ≐ lineSegpApB_value D

    givenProof : ⊢ ((<pA, pB>)  (<pB, pC>)) O = perp_axiom1 anglepApBpC D
M
```

Figure 11: MMT Situation Theory

```
namespace http://cds.omdoc.org/FrameIT M

view situation_problem_view : ?problem_theory -> ?situation_theory =
    Pa = pA D
    Pb = pB D
    Pc = pC D
    anglePaPbPc_value = anglepApBpC_value D
    anglePaPbPc = anglepApBpC D
    anglePcPaPb_value = anglepCpApB_value D
    anglePcPaPb = anglepCpApB D
    lineSegPaPb_value = lineSegpApB_value D
    lineSegPaPb = lineSegpApB D
    proof = givenProof D
M
```

Figure 12: MMT Problem Situation Theory View

5.2 Game (Unreal Engine)

After describing the MMT part of implementation, we will now continue by explaining the implementation of the game itself. This section focuses solely on the parts of the implementation relevant to the **FrameIT** method and not on the auxiliary parts such as the creation of the landscape.

In our game the problem is directly presented to the user by displaying a description of the problem. This representation of the problem in the game world can be seen in the screenshot shown in

Figure 3. After having received the problem, the user can explore the game world by walking through it and looking around. Through this exploration the user then can obtain a good understanding of the problem and think about possible solutions. For instance, the user is able to observe that the tree is straight and situated on flat ground surrounded by hills.

To solve the problem the user has access to a variety of tools, which allow him or her to discover new facts about the world. In order to store these facts on the game side, I implemented a set of classes that represent the different kinds of facts a user can discover. Each of these classes contains methods that can serialize the fact, the respective class represents, 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 user has access to different tools to discover new facts about the world. His or her most important tool is the *point marking tool* as he or she can use it to create a point fact by simply marking a location. This *point marking tool* 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 13 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.

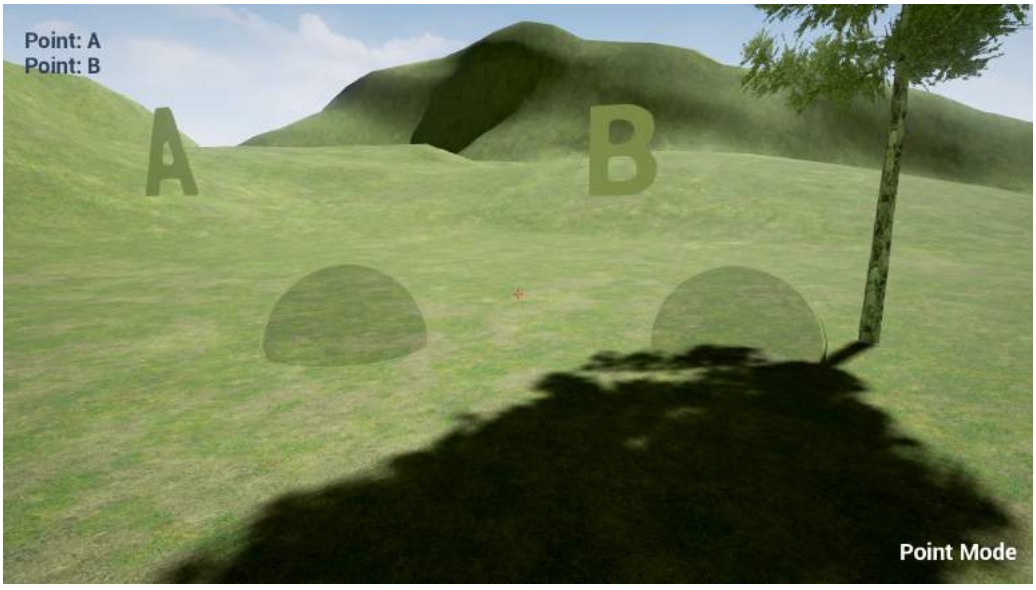

Figure 13: Point Marking Tool

Other important tools are the *distance measuring tool* and the *angle measuring tool*. With these tools the user can select two or respectively three points and thereby measure the distance or angle between them. 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. The Figures 14 and 15 show the result of applying these tools to a set of points. As one can observe from these figures, 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.

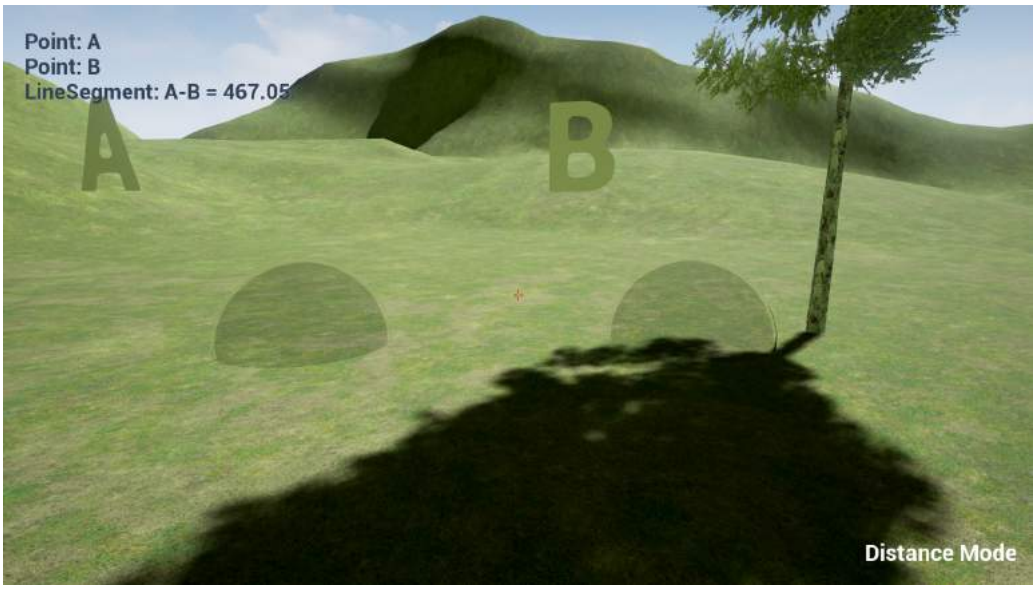

Figure 14: Distance Measuring Tool

Figure 15: Angle Measuring Tool

The first step in actually solving the problem is to decide on an approach. In our game the user can look through all available scrolls and select the one he or she wants to use. After the user decides

on a scroll he or she can measure the facts required by it. As one can see in Figure 16 by measuring facts about the world the user's fact list grows gradually.

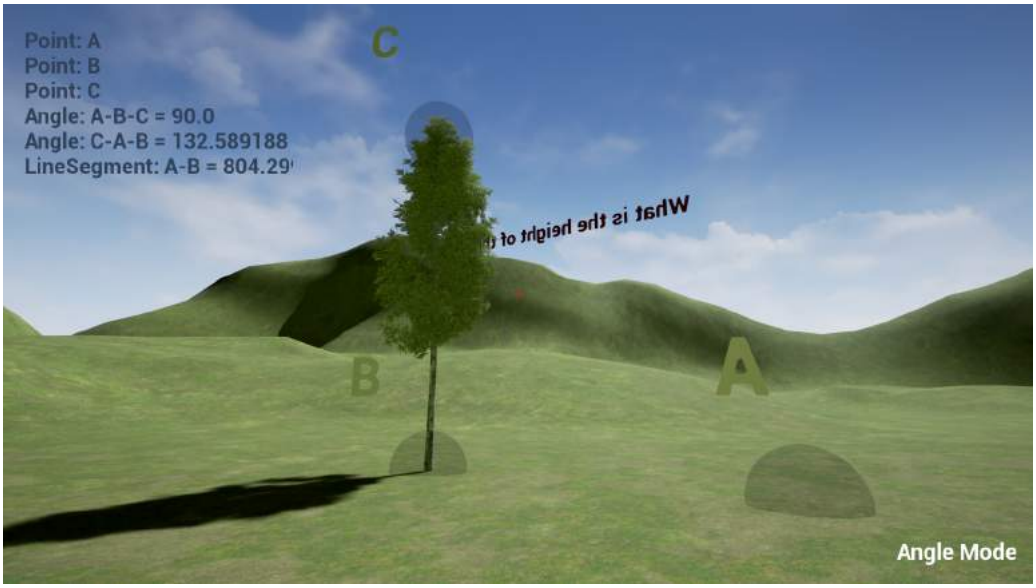

Figure 16: Measuring Facts about the World

After measuring all facts the user can enter the *View Mode* in which he or she assigns for each fact required by the scroll a fact from the fact list. This process can be seen in Figure 17. 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 an OMDoc document representing a view and saved as a file into the content folder of MMT. At the same time the user's fact list is serialized into an OMDoc situation theory and saved as a file into the same content folder. Next, a new MMT process is spawned that executes a specific FrameIT script, 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 18. 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.

It is important to mention that the scroll currently used in our game is implemented directly in code. The reason for this is that we are not yet dynamically generating scrolls from the problem/solution pairs. As soon as this generation is implemented our system would need to be adapted.

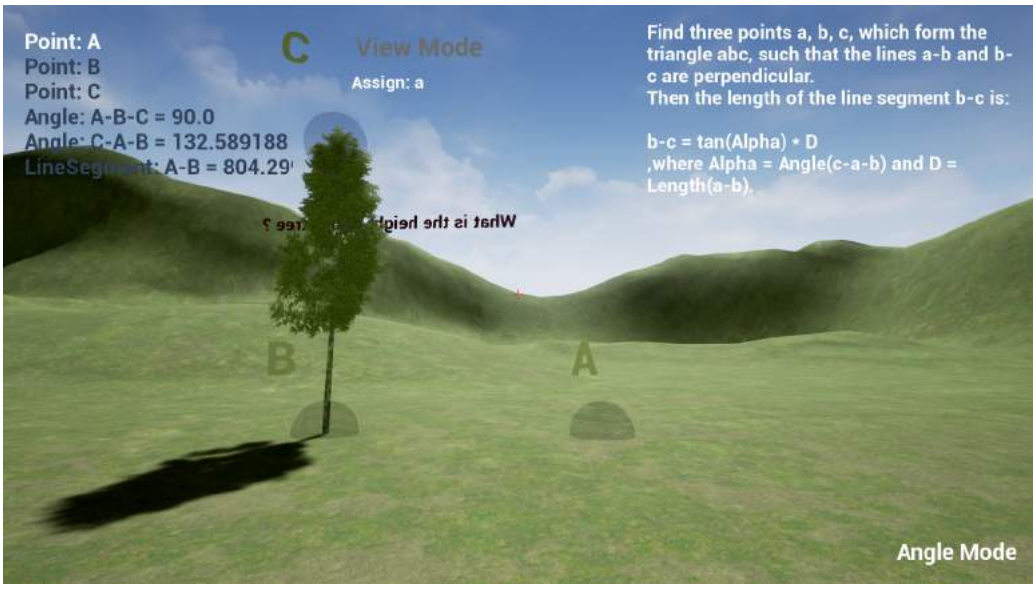

Figure 17: View Assignment

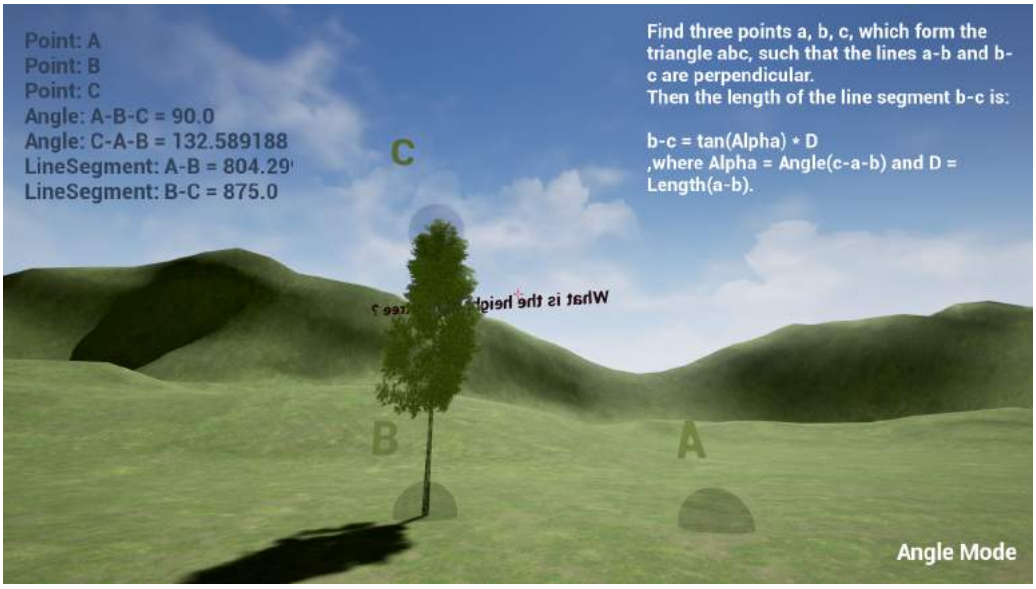

Figure 18: Scroll Result

6 Evaluation

As shown in the design and implementation section, our implementation actually works, which shows that our envisioned method could be used for creating full-fledged serious math games. Nevertheless, our system still requires several improvements before it can be used as the basis of more sophisticated educational games.

One obvious shortcoming of our serious game is the lack of content. At the moment the user has only access to one scroll and therefore is only able to solve one kind of problems. To fully evaluate our method and to demonstrate its usefulness, we would need to develop the serious game further

by implementing more scrolls and problems. Furthermore, this lack of content does not yet allow us to demonstrate the ability of the **FrameIT** method to support compound problems. Nevertheless, it is very clear from the theoretical work on the method that it is indeed able to support compound problems natively.

Given the lack of game content and the short time frame of this research project, I was unable to qualitatively and quantitatively analyze the usefulness and usability of the developed serious math game. However, as the method requires users to solve real world problems, one can assume that by playing the game, the user is gradually improving his or her problem solving skills.

Using the Unreal Engine was an excellent choice as it clearly demonstrated that the method could be used and implemented in a real game engine. Unfortunately, the connection between game engine and MMT is not ideal as the current method of exchanging information over the file system is not scalable. However, both the Unreal Engine and MMT have modules for HTTP based communication which could be utilized to exchange information more efficiently between the systems. Ideally, we would develop a completely semantic game engine that includes the required functionality provided by MMT directly. Unfortunately, creating such a game engine is an extremely expensive endeavor and will therefore probably not be developed in the near future. This is why connecting a real game engine to MMT as efficiently as possible seems like a good compromise.

7 Future Work

While the refined FrameIT method, with its proof of concept implementation, is a first step towards our goal of creating serious math games from logic we still need to research and develop important missing pieces. Future work regarding this topic can be roughly categorized into four areas.

The first area is future work on the FrameIT method itself, as it needs to be further refined and extended. The primary objective here would be to research and implement the generation of scrolls as discussed in section 4.3. Additionally, it is necessary to create more different problem/solution pairs in order for us to demonstrate the application of the method to more complicated problems. 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.

The second area of future work should 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 [DJKK12] as its basis and adapt it for the Unreal Engine.

Lastly, a qualitative and quantitative analysis of our serious game should be conducted to test if it actually helps students improve their problem solving skills. The results of such a study might reveal new areas of improvement which could be addressed by future research. Obviously, in order to effectively conduct experiments and studies we need to extended our proof of concept implementation to a full-fledged serious game.

8 Conclusion

In summary, I devised the first complete implementation of the **FrameIT** method and refined and extended the method itself during this research project. While our serious math game has currently many limitations one can already demonstrate its capacity as a tool for improving problem solving skills. Moreover, this first implementation revealed many areas of future work, such as refining the method itself or integrating MMT directly into the Unreal Engine.

To conclude, this project lays the groundwork for a new generation of serious games in education, which are based on logic and the **FrameIT** method, that will improve the ability of students to apply their obtained knowledge to real world problems. Thus, allowing them to become better at solving problems and thereby more effective in their future professions.

9 References

[Aca16] Khan Academy. Khan academy. https://www.khanacademy.org/, 2016.

[Co.16] Imaginary Number Co. Mathbreakers. https://www.mathbreakers.com/, 2016.

[DDKN11] Sebastian Deterding, Dan Dixon, Rilla Khaled, and Lennart Nacke. From game design elements to gamefulness: Defining "gamification". In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, pages 9–15, New York, NY, USA, 2011. ACM.

[DJKK12] Catalin David, Constantin Jucovschi, Andrea Kohlhase, and Michael Kohlhase. *Intelligent Computer Mathematics: 11th International Conference, AISC 2012, 19th Symposium, Calculemus 2012, 5th International Workshop, DML 2012, 11th International Conference, MKM 2012, Systems and Projects, Held as Part of CICM 2012, Bremen, Germany, July 8-13, 2012. Proceedings*, chapter Semantic Alliance: A Framework for Semantic Allies, pages 49–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[EG16] Inc. Epic Games. Unreal Engine 4. https://www.unrealengine.com/, 2016.

[KK11] Andrea Kohlhase and Michael Kohlhase. A naive storyboard for simsalabim. 2011.

[KK12] Andrea Kohlhase and Michael Kohlhase. Frames: Active examples for technical documents. 2012.

[Koh06] Michael Kohlhase. OMDoc – *An open markup format for mathematical documents [Version 1.2]*. Number 4180 in LNAI. Springer Verlag, August 2006.

[Koh08] Michael Kohlhase. Using LaTeX as a semantic markup format. *Mathematics in Computer Science*, 2(2):279–304, 2008.

[Koh14] Michael Kohlhase. Mathematical knowledge management: Transcending the one-brain-barrier with theory graphs. *EMS Newsletter*, pages 22–27, June 2014.

[MGLU09] Erica Melis, Giorgi Goguadze, Paul Libbrecht, and Carsten Ullrich. Activemath–a learning platform with semantic web features. *Semantic Web technologies for e-Learning*, pages 159–177, 2009.

[MS04] Erica Melis and Jörg Siekmann. *Artificial Intelligence and Soft Computing - ICAISC 2004: 7th International Conference, Zakopane, Poland, June 7-11, 2004. Proceedings*, chapter ActiveMath: An Intelligent Tutoring System for Mathematics, pages 91–101. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[Rab] Florian Rabe. Theory expressions (a survey).

[Rab13] Florian Rabe. The MMT API: A generic MKM system. *CoRR*, abs/1306.3199, 2013.

[Rab15] Florian Rabe. The future of logic: Foundation-independence. *Logica Universalis*, pages 1–20, 2015.

[Rac12] Daniel Rachev. FrameIT:an user interface for applying mathematical theories to real world problems. Bachelor thesis, Jacobs University Bremen, October 2012.

[Zyd05] M. Zyda. From visual simulation to virtual reality to games. *Computer*, 38(9):25–32, Sept 2005.