FrameIT: Detangling Knowledge Management from Game Design in Serious Games

Michael Kohlhase[0000−0002−9859−6337]1, Benjamin Bösl1, Richard Marcus[0000−0002−6601−6457]1, Dennis Müller1, Denis Rochau, Navid Roux[0000−0002−8348−2441]1, John Schihada1, and Marc Stamminger1

1Computer Science, FAU Erlangen-Nürnberg, Erlangen, 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 connecting the Mmt API with a state-of-the-art game engine.

1 Introduction

Serious games could be a 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]. Beyond educational games for students, the term "Serious Game" is used for games that help to acquire skills in general. This includes training professionals of basically all industry sectors.

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 for complex subjects like science, technology, engineering, and mathematics (STEM) are currently very complex, domain-specific, and expensive even though their motivational effects could be disruptive right in these areas. Even more seriously, developers of such games need to combine the skill sets of game development, pedagogy, and domain expertise, a rare combination indeed.

To alleviate this, we propose the **FrameIT Method**, which – instead of using ad-hoc methods for dealing with the underlying STEM domain knowledge in the game – uses established mathematical knowledge management (MKM) techniques and implementations. It loosely couples a game engine for interacting with virtual worlds with the MMT system, which performs knowledge representation and management services, thus detangling the domain knowledge integration from the game development process. The main mechanism involved is the maintenance of a mapping between objects of the virtual world and their properties ("facts"), which are formally represented in OMDoc/MMT. On this basis, learning objects in the form of represented theorem statements can *i*) be visualized in the game world ("scrolls") for the player to understand, *ii*) be instantiated by the player by assigning a game object to every required assumption, and *iii*) can, together with their instantiations, be represented in MMT as OMDoc/MMT views. The latter enables validity checking and computation of results which can then be transferred back into the game world bringing things full circle.

The FrameIT Method is supposed to increase a player's understanding of formulae by making them apply such abstract formulae in concrete settings happening within a game world. To fulfill a formula's assumptions, the player has to perform a combination of selecting, moving, and generating game objects. With the help of OMDoc/MMT in the background and back-and-forth synchronization, concrete outcomes of formula applications can immediately be visualized for the user in the game world, too.

The Tree Example At this point, we would like to introduce a running example of an in-game word problem for a serious game. We use this problem in our serious game prototype as well throughout this document to progressively explain the FrameIT Method.

Concretely, the player is presented a tree in a forested 3D world and is asked to determine its height using a limited set of **gadgets**; each of those providing **facts** about the world, e.g. acquirable angles and lengths from the player's perspective (cf. Figure 1). The intended solution is to frame this problem in the language of trigonometry as finding the length of the opposite side given an angle and the adjacent side. Other solutions are also possible, e.g. choosing an isosceles 45°-45°-90° triangle, for which both legs of the triangle then have the same length.

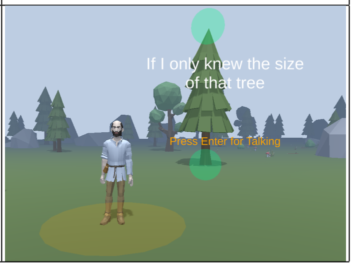
Word Problem	**Game Problem**
How can you measure the height of a tree you cannot climb, when you only have a laser angle finder and a tape measure at hand?	 If I only knew the size of that tree Press Enter for Talking

Fig. 1: Example Problem

Didactically, the game world is rigged so that the gadgets produce only facts acquirable from the player's perspective. For instance, they cannot climb the tree, and hence the provided measuring tape gadget disallows measuring the tree's height. Instead, the user is expected to use **scrolls**[1] to discover new facts about the world in alternative ways. In the problem at hand, such a scroll on trigonometry could provide the length of the opposite side of a right-angled triangle given an angle and the length of an adjacent side – both of which are acquirable from the player's perspective.

Contribution In this paper, we present the FrameIT Method as a new approach to knowledge management in serious games and an implementation of this, the UFrameIT system. We have implemented all system components and developed APIs that allow an integration of the MMT system with Unity, a state-of-the-art game engine. With the new framework, building a serious game should be a matter of formalizing the background knowledge in OMDoc/MMT and providing the necessary gadgets and scrolls. We confirm this hypothesis by instantiating UFrameIT into a very simple serious game: FrameWorld-1. The project is available on GitHub, we provide a demo video and a playable prototype; all of these can be found at https://uframeit.github.io/.

Related Work We limit ourselves to describing how our primary contribution, the FrameIT Method, fits into the spectrum of existing methods of knowledge management in (educational) games and tools. The concerns of knowledge management and actual game realization in source code can be completely intermingled. This is especially the case in games that are built from the ground up. Going noticeably further, we can find games employing dedicated physics engines and frameworks for handling user objects. One noteworthy library of such small-scale yet modular games for STEM education are the PhET Interactive Simulations [PhET]. Continuing on the spectrum, we can identify domain-specific languages being increasingly used. A well-known example of this is GeoGebra [GG], a graphics calculator employing a dedicated computer algebra system. The concept of explicit knowledge integration can also be found in state-of-the-art game engines. There, various forms of dataflow programming are used, for instance, to construct graphs to specify shader materials, visual effects, animation transitions, and even gameplay interactions. Finally, instead of DSLs we can also use a dedicated MKM system (our approach), which gives us the most flexibility in knowledge formalization.

Overview In the next section, we mainly recap previous work on OMDoc/MMT. We then continue describing our approach in a progressive, threefold way. In Section 3, we first describe the FrameIT Method from a purely conceptional viewpoint. Then, instantiating that concept, we present and discuss our implemented framework UFrameIT in Section 4. Finally, in Section 5 we show our realization of the running example within UFrameIT. In Section 6, we give a short conceptual evaluation of the FrameIT Method and conclude the paper in Section 7.

[1] The name "scroll" is meant to evoke the fact that the knowledge contained in it is a valuable commodity in the game.

Acknowledgements The development of the FrameIT Method has profited from discussions in the KWARC group, in particular from contributions by Mihnea Iancu and Andrea Kohlhase. The implementation reported on in this paper completely re-implements [RKM16] and extends it significantly.

2 Preliminaries

For the concept and implementation of the FrameIT Method, we require an MKM system capable of storing, relating, and combining knowledge items in a structured knowledge graph. To the best of our knowledge, besides the MMT system, the only other systems supporting this sufficiently are Hets [MML07] and Specware [SPEC].

2.1 Learning Object Graphs as OMDoc/MMT Theories

In this work, we choose the OMDoc/MMT language [RK13] as a fitting theoretical framework together with its reference implementation in the MMT system [Rab13], which is a general foundation- and logic-independent framework for creating formal systems [MR19]. Below, we briefly recap the language in a way suited for our applications in this paper. Nonetheless, our methods are agnostic on the specific choice of an MKM system as long as it supports the features elaborated on below in some way. Indeed, we reflect the loose coupling of our approach in the structure of this paper by having Section 3 detailing the FrameIT Method without assumption of any implementation details.

Storing OMDoc/MMT organizes knowledge into theories and relates theories via views. A **theory** is essentially a list of typed constant declarations of the form $c\colon E\ [= e]\ [\#\ N]$. Here, c is the (theory-local) identifier, the well-typed expressions E and e the type and definiens, and N some notation. Expressions are well-formed terms over all previous declarations in scope. By leveraging a suitable foundation and logic as well as the Curry-Howard correspondence, we can represent a wide range of formal knowledge including type, function, and predicate symbols, axioms, theorems, judgements, and inference rules. Moreover, for structuring purposes, theories can include (import) knowledge of other theories. Since inclusions are a special case of views, we suggestively write $S \hookrightarrow T$ for a theory S being included in a theory T.

Relating In general, two theories S and T can be related by a **view** $v\colon S \rightsquigarrow T$, which is a well-typed map mapping every declaration in S to a T-expression. Views can be thought of as refinements from some abstract theory S to a more concrete theory T. For instance, in our running example we could represent a theory of triangles in S, a concrete game world set up by the player in T, and utilize a view $S \rightsquigarrow T$ to understand the tree and the shadow cast by it as forming a triangle. In particular, as a result of well-typedness we get *truth preservation* as a metatheorem: under a view, the images of axioms/theorems and proofs in the domain theory are again theorems and proofs in the codomain theory. In the context of our example, this enables instantiating abstract theorems, such as trigonometric identities, in the concrete world, e.g. to compute the tree's height by only knowing the shadow's (projected) width and the enclosed angle.

A **theory graph** is a multigraph emerging from a collection of theories and views together. We will use theory graphs as **learning object graphs** in the FrameIT context. They 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.

Combining The final feature we require from an MKM system for our purposes is to *combine* knowledge. In the OMDoc/Mmt language, we can phrase this as computing pushouts in the category of theories and views. In this category, pushouts along inclusions always exist; see Figure 2 for the general scheme. Intuitively, the pushout P is formed as the union of S and T such that they exactly share R. In the FrameIT Method, we make extensive use of pushouts as a way to translate abstract conclusions in T into the context of a concrete situation in S. This is legitimized by first constructing the upper view v in Figure 2, which serves to *frame* parts of S as the abstract preconditions stored in R; hence the name FrameIT.

$$\begin{tikzcd}
R \arrow[r, squiggly, "v"] \arrow[d, hook] & S \arrow[d, hook] \\
T \arrow[r, squiggly, "s"] \arrow[ur, phantom, "\lrcorner" near end] & P
\end{tikzcd}$$

Fig. 2: Pushout

2.2 Unity: a Multi-Platform Game Engine

Since our goal is the development of a knowledge-based engine for serious math games, we encounter the need for a correspondent graphics engine twice: once, to create a system that can interoperate with the MKM system and, once, to actually implement a game prototype using this framework. To cope with this, we use the Unity game engine [Uni]. As an industry standard with a big community – providing materials, assets and tutorials – it meets all our requirements. While it is easy to learn the basics, it is yet a powerful and flexible tool, supporting deployment to basically every platform, including VR and AR devices. It greatly reduces the amount of effort to create virtual worlds by largely taking care of rendering as part of its huge API to implement game interactions and interfaces. In particular, it also offers an interface for communicating with a *RESTful* API, which we exploit in our implementation to interact with the Mmt system.

3 The FrameIT Method

We propose that – at least for the domain of mathematical knowledge – serious games be implemented with a dedicated MKM system in the background leveraged for storing, relating, and combining knowledge. In our concept, we exploit features provided by the MKM system and expose them to the player by means of appropriate user interfaces such that players can easily explore, play, compute, and verify solutions to in-game puzzles. From the many conceivable kinds of applications, in this work we focus on the task of **framing puzzles**. Such puzzles challenge the user to frame concrete tasks in the 3D game world, such as measuring a tree's height, as abstract problems, such as finding an opposite's length in trigonometry.

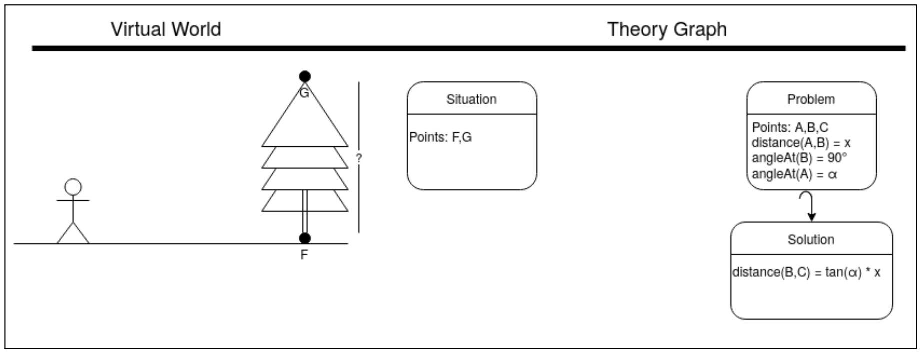


Fig. 3: The FrameIT Method as a Process – Initializiation

3.1 Exemplary Playflow

The main contribution of the FrameIT Method is the division of labor between game engine and MKM system, which offers several advantages regarding development workflows and knowledge management. To get a better intuition of the method, we will go through the process of solving our tree example step-by-step showing what goes on in both subsystems (on the left and right of the Figures 3 and 5 to 7). This also allows us to introduce the pertinent concepts by way of our running example.

Initially in the game, the user is presented our word problem together with some initial "background knowledge" they are allowed to apply throughout solving the puzzle (cf. Figure 3). This background knowledge encompasses *facts* and *scrolls*: **Facts** are typed and arbitrarily complex knowledge items. For example, labelled 3D points marked in the world, such as $A := (1, 0, 0)$, can be facts. They can originate from multiple sources including level-dependent background knowledge and in-game exploration by the player themself. In our example, the user initially gets two point facts (namely F and G) marking the tree's endpoints. As is the case with all facts, they are kept synchronized with the knowledge side, which we can observe in Figure 3 as declarations in the **situation theory**. This theory is a designated, possibly level-dependent theory encompassing the world knowledge provided or gained so far.

Scrolls complement the concept of facts via a mechanism to obtain new facts from existing ones – much like mathematical theorems. The game provides the user with the OppositeLen scroll (see Figure 4), which operationalizes the mathematical theorem the game wants to teach. Namely, it requires three point facts a, b, c, the angle $\measuredangle cab$, and the knowledge of $\measuredangle abc = 90°$ as input and in return provides an identity about $|\overline{bc}|$. This way, we see that scrolls can serve us as **learning objects** in serious games. On the knowledge side, we can represent them as **Problem/Solution theory pairs** (cf. Figure 5), where the problem theory encapsulates the scroll's universal variables and preconditions, and the solution theory contains the desired assertions (results) in context of the former. In the process below, we will see that theorem application then becomes pushout computation in our sense.

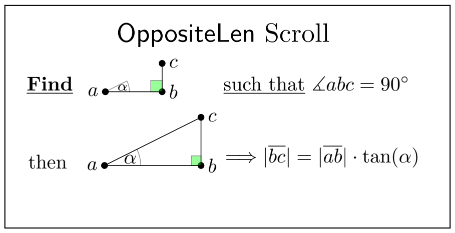

OppositeLen Scroll
Find
a
α
b
c
such that $\measuredangle abc = 90°$
then
$\Longrightarrow |\overline{bc}| = |\overline{ab}| \cdot \tan(\alpha)$

Fig. 4: The OppositeLen Scroll

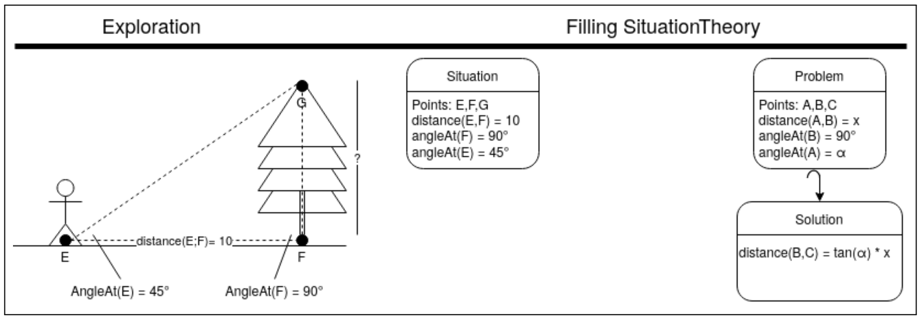


Fig. 5: The FrameIT Method as a Process – Step 2

In the second step, the user explores the virtual world and experiments with the given facts and scrolls. In some serious games, this happens off-band by the player with pen and paper. By contrast, the FrameIT Method actively encourages in-game exploration and even requires it to solve puzzles. World exploration can involve marking new points and lines within the world, possibly guided by scrolls like the OppositeLen scroll our player has been presented. Concretely, we imagine they use the **pointer gadget** in the game UI to mark a point E on the ground and the **line gadget** to mark a triangle through E and the tree's endpoints. Moreover, they measure $\measuredangle GEF = 45°$ and $\measuredangle EFG = 90°$ using some protractor gadget. On the side of the MKM system in Figure 5, we see that the collected facts are communicated to the MKM system as soon as they are created: the situation theory grows.

In the third step, the player **frames** the in-game word problem in terms of the OppositeLen scroll by mapping every scroll input to a game world object. Here, the inputs for the point facts a, b, c, the enclosed angle $\measuredangle cab$, and the right angle $\measuredangle abc$ are mapped to the facts E, F, G, $\measuredangle GEF = 45°$, and $\measuredangle EFG = 90°$, respectively. This assignment is communicated to the MKM system which

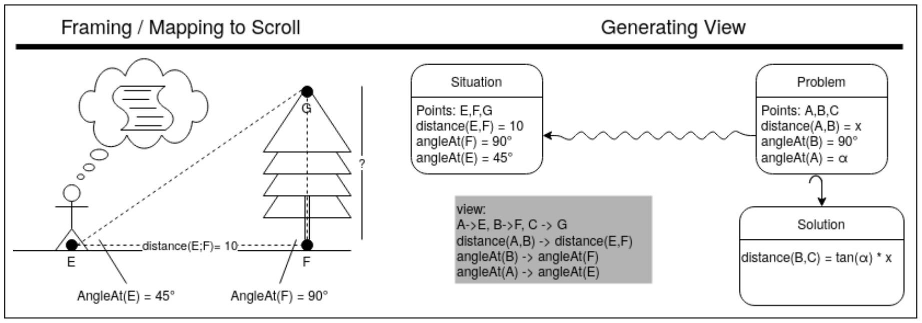


Fig. 6: The FrameIT Method as a Process – Step 3

establishes that it constitutes a view – we call it the **application view**. Critically for our serious game use case, it establishes the precondition that $\triangle abc$ is a right-angled triangle which justifies the application of the OppositeLen scroll. If the player frames the game problem with an assignment that does *not* lead to a view – e.g. if the ground the tree stands on is sloped and thus the angle $\measuredangle EFG$ is different from 90° – the MKM system will reject the framing and can pinpoint exactly where the error lies.

In the final step (cf. Figure 7), the MKM system computes the pushout of the application view over the inclusion of the problem into the solution theory. Moreover, it simplifies terms, computes values, and reports to the game engine that the user has solved the puzzle. Concretely, success was determined by checking whether the fact $|\overline{FG}|$ simplifies to a numeric value in context of the pushout theory. This formal notion corresponds to the intuitive puzzle objective of finding that length.

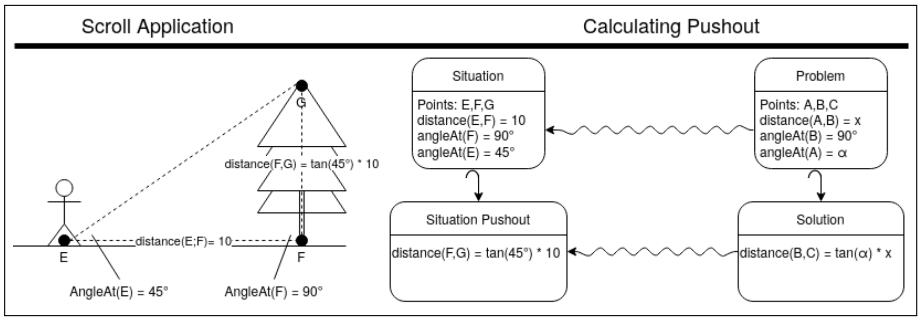


Fig. 7: The FrameIT Method as a Process – Step 4

Having solved the puzzle, the player can now proceed to choose a new puzzle to play. Importantly, the knowledge gained so far is *not* thrown away, but kept for future use by the player. For example, in subsequent puzzles the player can use the tree's height as input for other scrolls. This effect is easily achieved by updating the pointer to the situation theory to the computed pushout theory in the course of the last step.

Playing in Practice Note that we presented an idealistic chronological order for simplicity only. In general, players might do several steps simultaneously, make mistakes in framing, and repeat previous steps. Moreover, levels might come bundled with multiple scroll libraries for the user to apply and choose from. All in all, much in the spirit of a working mathematician, the tasks of exploration, scroll application, and success are blurred in practice. See Section 5 for a realization of a game that allows to do all of these.

3.2 Acquiring Facts and Using Scrolls

Facts are a central part of the FrameIT Method. In our running example, we have so far only seen facts being acquired by marking/measuring things in the 3D world and by scroll application. Below, we give an extended, though non-exhaustive, compilation of ways to acquire facts.

- **Exploration of the 3D World:** Players can explore the 3D world by means of **gadgets**, which are a mechanism in the game UI to mark or measure things of interest in the game world. Our implementation includes gadgets to mark points, lines, angles, and distances among others. Upon usage, all these gadgets generate facts.
- **Scroll Application:** Successful scroll application leads to one or more facts being output.
- **Discovery, Awards, and Trade:** Serious games could be designed such that a player can stumble upon and discover facts within the world, e.g. by "talking" to non-player characters. Moreover, in more elaborate story designs, levels may come with a prize fact to earn upon success, which is then required for subsequent levels. Finally, assuming some kind of multiplayer mode, we might also allow players to share and trade facts.

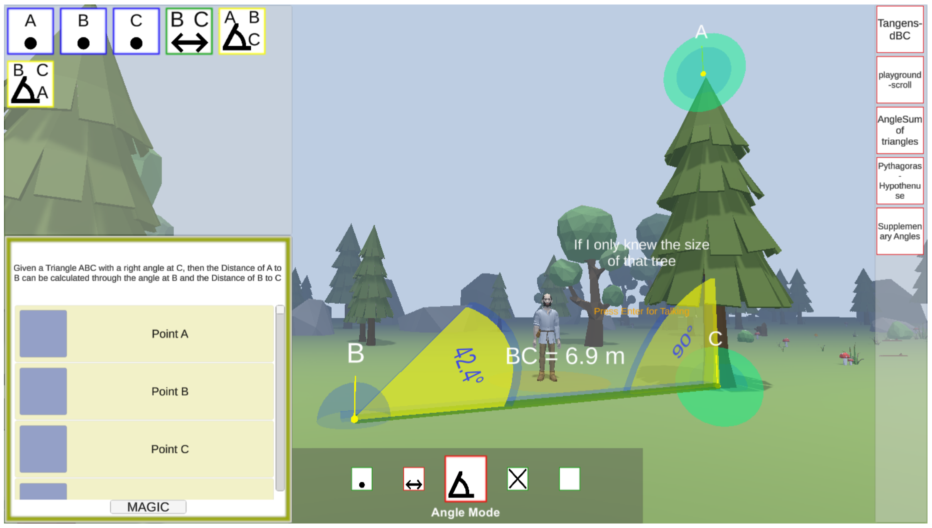
Common to all ways of obtaining facts is that upon acquisition they are synchronized with the MKM system. Namely, it is supposed to serve as a single source of truth for all knowledge items. We will discuss the implementation of an appropriate framework next.

4 The UFrameIT Framework (Implementation)

We have implemented the FrameIT Method as a prototypical serious game framework we call UFrameIT. Concretely, we extend the existing Mmt system with an interface for incremental fact synchronization and implement a general infrastructure for fact management, gadgets, scrolls, and framing in the C#-based API of the Unity game engine.

We have also instantiated the UFrameIT framework with a simple proof-of-concept game FrameWorld-1, which we describe in the next section. We separate the two concerns – even though they were developed together – to give an intuition of the relative efforts.

Figure 8 shows the main parts of the UFrameIT framework: the environment, the first-person player, the problem definition, as well as facts and scrolls at work in FrameWorld-1. It also shows the **Framing UI**, which allows **framing** and tool selection. We describe this in detail below. Lastly, in the middle we can see the laser angle finder **gadget** at work after it has been selected from the gadget bar at the bottom of the screen.



Given a Triangle ABC with a right angle at C, then the Distance of A to B can be calculated through the angle at B and the Distance of B to C

Fig. 8: Measuring Facts about the World

4.1 Extending Unity with Facts, Scrolls, Gadgets, and Framing

To incorporate the FrameIT Method we mainly need two things: gadgets and an interactive user interface.

Gadgets and Facts On a technical level, gadgets consist of the following parts:

- To identify tools within the game, they need **graphical representations**. Currently, we only use a planar icon for the UI, but in the future, we plan to have 3D objects to show the gadgets in the virtual world.
- The activation of a gadget triggers **gadget events** that initialize or update its internal state. These events are used for communication between the player and the gadget.
- Gadgets give feedback to the player via **gadget visual effects**, e.g. for showing assisting previews during fact creation.
- Finally, gadgets trigger **fact events** to initiate the creation of the appropriate facts.

Facts are managed by MMT but, just as gadgets, they have graphical components: a Unity GameObject for interaction in the virtual world and an icon for interaction in the UI.

In order to develop a new gadget, there are three main modules which have to be extended: FactManager, FactSpawner and VisualEffectsManager. These modules cope with the different gadget parts described above. The FactManager is aware of the currently active gadget and handles the gadget-specific inputs made by the player. If necessary, it delegates work to the other modules. For instance, when a gadget was used successfully, it updates the global fact list (by addition or removal) and triggers the FactSpawner to arrange for the facts' in-game visualization. Moreover, for visualizing assisting previews in the course of using a gadget, the FactManager delegates and transmits the necessary data to the VisualEffectsManager. All of the modules assume that suitable fact types and gadgets producing instances of them have previously been established. Additionally, every fact type needs to be given a formalized counterpart on the MMT side. Hence, if a new gadget exceeds the current range of functionality, these parts may also need adaptation.

Framing UI On the lower edge of the screen, players can find the **Gadget Toolbar**, which allows access and activation of the respective gadgets. To interact with the measured facts, the user can activate an overlay that freezes the underlying game and gives access to framing (cf. Figure 8). **Facts** are depicted as small tiles and are collected in the fact inventory on the top left. Complementarily, available **scrolls** are shown on the right edge, of which the currently active scroll is shown beneath the fact inventory. Players can then fill the scrolls with facts via drag & drop. When the player clicks the "Magic" button, UFrameIT constructs and transfers the application view to MMT, which computes the pushout and, after successful verification, hands back the resulting facts.

4.2 Communication

To allow MMT to process information and give feedback according to the FrameIT Method, we use a very fine-grained communication approach. The backend server provides a RESTful-interface with endpoints to add facts (one endpoint per fact type), generate views, request pushout computations, and to list available scrolls. The corresponding payloads are transmitted in the JSON data format. There are three different types of events that trigger communication with the server:

- **Game World Triggers:** These automatically send requests during interaction with the game world but are not used for our simple example.
- **Fact List Modification:** We report all changes to the fact list to the server. Most prominently, these changes are triggered by gadgets. Each gadget-generated fact entails sending an HTTP request including the fact details to the server. On the MMT side, the putative fact is first checked for validity, then, upon success, stored as corresponding declaration(s) in the situation theory, and lastly, its generated declaration identifier is sent back to Unity.
- **Attempt of Scroll Application:** When the player tries to apply a scroll, a test for applicability is started: The mappings of the filled scroll are sent to the server and packaged into a putative view by MMT. The latter is then run through the type checker, whose outcome is reported back to the game

engine. Upon success, the game engine requests the pushout computation wrt. the Problem/Solution theory pair representing the current scroll and updates the UI with the results.

5 FrameWorld-1: A Simple Serious Game in UFrameIT

FrameWorld-1 is a simple game instantiating the UFrameIT framework into a proof-of-concept game that is inspired by our running example and the playflow from Section 3.1. As most of the infrastructure comes from UFrameIT, the only "game contents" we had to develop for FrameWorld-1 were the game world, the problem-specific gadgets, a formalization of the background theory of 3D geometry and trigonometry, and appropriate scrolls.

5.1 A Simple Virtual World

To build a game, we require a world for the player to explore. With Unity, we simply added an object serving as ground together with the default first-person camera asset for navigation. To simplify the process of applying basic geometry, we kept the ground of the world completely flat for our first game. To bring the scene to life, we populated the scene with assets that are freely available at the Unity asset store.

Gadgets and Facts in FrameWorld-1 Gadgets are the core way of interacting with the world; for FrameWorld-1 we created three gadgets. The **pointer gadget** marks a point in the game world and produces a new fact that declares a labeled point. Upon gadget activation, objects in the environment that shouldn't be markable, e.g. the sky or existing points, are set to be ignored. Moreover, **snap zones** are activated. Placing a point within these zones positions it exactly at the center of the zone, which is necessary to accurately mark the root and the top of the tree. The user can then relate two or three different points by measuring the distance between them with a **measuring tape gadget** or the angle between them with the **laser angle finder**. An angle is defined by the selection of three existing point objects. Every single selection triggers an event that updates the internal state of the gadget. After the second point is selected, we preview the angle by following the mouse pointer until the third point has been fixed (cf. Figure 9). Distance measuring is implemented analogously, in this case, with a preview line following the cursor. Importantly, we let the line only follow the cursor up to the height of the player and prevent connection with points which are higher than that. Even though these three gadgets were developed for FrameWorld-1, it is clear that they are generally useful for problems based on 3D geometry and can thus be shared with subsequent games.

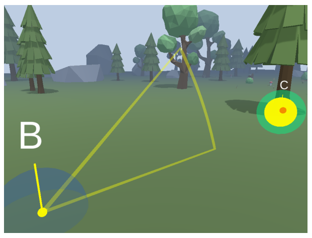


Fig. 9: Measuring Angles

Playing the Game The player automatically obtains a set of scrolls by starting FrameWorld-1. They learn about the puzzle they need to solve by talking to the non-player character and by subsequent exploration of the world by the means of the FrameIT Method. Delivering the height of the tree to the character talked to completes the game and triggers some visual feedback indicating success – the fireworks in Figure 10.

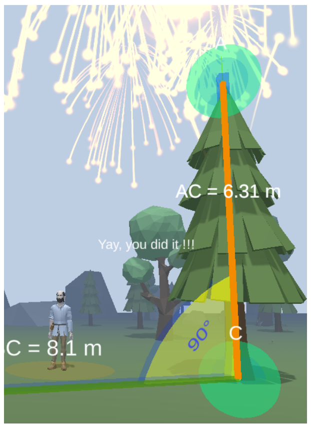


Fig. 10: Success

5.2 Domain Knowledge and Scrolls

For FrameWorld-1, we have extended the MitM Ontology [MitM] by seven theories for 3D geometry and trigonometry, which can be found in [UFM]. We decided on 3D instead of planar geometry as the virtual world is 3D and a mapping to its 2D variant would engender an additional transformation step that we would have to communicate to the player. Additionally, this allows us to implement more advanced 3D geometry scrolls in the future.

```
theory problem : ?geometry =
  meta scrollName "OppositeLen" ❘
  meta scrollDescription "Given a triangle ABC right-angled at C, the
    distance AB can be computed from the angle at B and the distance BC" ❘
  meta solutionTheory ?tan_solution ❘

  A: Vec3 ❘ B: Vec3 ❘ C: Vec3 ❘ // triangle's endpoints ❙
  distBC_val: ℝ ❘
  distBC: DistFact B C distBC_val ❘
  angleABC_val: ℝ ❘
  angleABC: AngleFact A B C angleABC_val ❘
  angleBCA: AngleFact B C A 90.0 ❘ // required 90.0° angle at BCA ❙
❚
theory solution : ?geometry =
  include ?problem ❘
  distCA_val: ℝ | = (tan angleABC_val) · distBC_val ❘
  tangentScroll: V ⟶ V ⟶ V ⟶ DistFact A C distCA_val ❘
  distCA: DistFact A C distCA_val | = tangentScroll A B C ❘
❚
```

Fig. 11: MMT Problem/Solution theory pair (modified for readability)

For the scroll in the running example, we use the Problem/Solution theory pair shown in Figure 11. The problem theory defines the required abstract situation of the scroll: a right-angled triangle, an angle, and the adjacent's length must all be known. In this context, the solution theory describes the scroll's output: the length of the opposite calculated via the tangent function. In the implementation we recognize and identify this pair by means of the meta annotations present in the problem theory.

Formalization in Detail Recall that scrolls may represent theorems and in those cases they should only be applicable on situations fulfilling the theorem's preconditions. Fortunately, we can leverage Mmt as an MKM system to enforce such conditions. For example, our background theory provides us a separate *distance type* for *every* real value of distance and two points. Using such a distance type for `distBC` in the problem theory allows us to enforce that only correct distance facts get mapped to it. For instance, a putative view mapping a distance fact for $|\overline{AC}|$ or even $|\overline{CB}|$ to `distBC` would lead to a typing error. We follow a similar approach for angle facts (cf. `angleABC` and `angleBCA`). Note that for `angleBCA` we fix the only correct angle of 90.0° directly in the type. In contrast, for the previous distance fact `distBC` and for `angleABC`, we used extra (unconstrained) declarations that make the actual value being mapped as the distance and the angle freely selectable. After all, these values are universally quantified over in the theorem statement.

Taking a step back from these practical experiences, we return to a conceptual level in the next section and evaluate the FrameIT Method.

6 Conceptual Evaluation

In the introduction, we have already given an account on how our approach fits on the spectrum of knowledge management in serious games. Now we evaluate it relative to two aspects in which we deviate from the other approaches.

First, we employ a **dedicated mechanism** for knowledge management instead of handling knowledge within source code. This is similar to GeoGebra, and in contrast to PhET Interactive Simulations which implements this concern in its JavaScript source code. The key benefits of doing so are:

- **Development Workflow Separation:** Traditionally, the game developer has to model complex problems within the game world, relying on frequent communication with experts to ensure that all aspects are implemented correctly. By encapsulating the knowledge integration, we can reduce the probability of mistakes during the knowledge transfer between domain experts and developers.
- **Reusing Knowledge:** As the (mathematical) background knowledge is independent from the game implementation itself, it does not rely on any specific programming language or game engine. This means that the knowledge formalization process only needs to happen once and different games can make use of it. Indeed OMDoc/Mmt has been designed to support knowledge re-use in practice.
- **Reusing Game Design:** This is dual to the point above. Given a sufficiently declarative implementation API of the game engine side, the FrameIT Method allows the game to be updated by simply adapting the theory graph.

Further advantages stem from using a very **modular and expressive system** like an MKM system. Again, GeoGebra which uses a computer algebra system heads into a similar direction as we do. On the other hand, approaches reimplementing such business logic in source code, such as PhET simulations, are arguably more flexible, but not necessarily modularly so. The following features can profitably be imported from an MKM system:

– **Dependency Handling**: The MKM system can be used to track formalized dependencies of game world objects that have been given a suitable counterpart on that side. Thus, after knowledge integration, developers can often avoid to reimplement these kinds of relation handling.
– **Feedback**: The MKM system can detect at which point a player's solution fails and to some extent also why. This allows to give feedback helping players to spot and rectify problems while solving puzzles.
– **Multiple Solutions**: With careful implementation of puzzle objectives in the MKM system, the game can be made agnostic to solution paths. Thus, if there are multiple ways to complete the game, the user is free to do so by default.
– **Compound Problems/Solutions**: By treating facts and puzzle objectives in a uniform way, we can naturally construct compound problems asking for facts to be obtained by subproblems. We have presented a simple example, but it is not difficult to think of more advanced examples that require multiple scroll applications.

Nonetheless, employing a separate MKM system also introduces potential issues. In more complex games the sheer number of communication requests might impact game performanc. Additionally, explicit modeling of background knowledge entails accounting for edge cases, which can be worked around in traditional (code-the-behavior) approaches.

7 Conclusion

We have presented a novel application of MKM technology: knowledge management in serious games, which we call the FrameIT Method. This principle alleviates the creation of games which, for instance, teach the application of simple mathematical models in geometry by instantiating them in virtual worlds. To realize the FrameIT Method, we have created an interface between the MMT system and Unity. This prototype implementation shows that combining a game engine with an MKM system is not only possible but indeed useful: The explicit representation of the underlying domain knowledge and the game world's situation in the MKM system allow for checking the applicability of the model on the MKM side. Consequently, our approach creates separated workflows and encourages reuse of content.

We have instantiated the UFrameIT framework to obtain FrameWorld-1, a simple serious game which challenges players to solve basic geometric problems using "scrolls" derived from 3D geometry and trigonometry. In accordance with our goals, our framework allowed to formalize knowledge in MMT largely independently from the remaining game development. Dually, we were also able to implement the game itself generically by building user-interface "gadgets", without necessitating domain expertise in geometry.

References

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

[GG] International GeoGebra Institute. *Graphing Calculator – GeoGebra*. URL: https://www.geogebra.org (visited on 05/27/2020).

[MitM] *MitM/core*. URL: https://gl.mathhub.info/MitM/core (visited on 01/18/2020).

[MML07] Till Mossakowski, Christian Maeder, and Klaus Lüttich. "The heterogeneous tool set, Hets". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2007, pp. 519–522.

[MR19] Dennis Müller and Florian Rabe. "Rapid Prototyping Formal Systems in MMT: 5 Case Studies". In: *LFMTP 2019*. Electronic Proceedings in Theoretical Computer Science (EPTCS), 2019. URL: https://kwarc.info/people/frabe/Research/MR_prototyping_19.pdf.

[PhET] University of Colorado. *PhET Interactive Simulations*. URL: https://phet.colorado.edu (visited on 05/27/2020).

[Rab13] Florian Rabe. "The MMT API: A Generic MKM System". In: *Intelligent Computer Mathematics*. Ed. by Jacques Carette et al. Lecture Notes in Computer Science 7961. Springer, 2013, pp. 339–343. DOI: 10.1007/978-3-642-39320-4.

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

[RKM16] Denis Rochau, Michael Kohlhase, and Dennis Müller. "FrameIT Reloaded: Serious Math Games from Modular Math Ontologies". In: *Intelligent Computer Mathematics – Work in Progress Papers*. Ed. by Michael Kohlhase et al. 2016. URL: http://ceur-ws.org/Vol-1785/W50.pdf.

[SPEC] Kestrel Institute. *The Specware System*. URL: https://www.kestrel.edu/home/projects/specware/index.html (visited on 05/27/2020).

[UFM] *Formalizations for UFrameIT FrameWorld*. URL: https://gl.mathhub.info/FrameIT/FrameWorld (visited on 03/19/2020).

[Uni] Unity Technologies. *Unity Realtime Development Platform*. https://unity.com/. Version 2019.3.6. URL: https://unity.com/ (visited on 03/19/2020).

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