

# The UFrameIT Project

Sebastian Weber

March 2021

## Abstract

Serious games are growing bigger and bigger but one sector seems to miss the opportunity to use them. In schools they are not as present as they could be. One reason for that might be that combining the theoretical and practical aspects of developing a serious game can be quite challenging.

The **UFrameIT** project deals with this problem and also helps teachers to develop their own serious games. This approach is based on the **FrameIT** method which tries to manage the underlying knowledge using *MMT theories*. In this paper I want to show the differences between the *UFrameIT* framework and other serious games as well as their advantages and disadvantages.

## 1 Introduction

*Motivation* Nowadays games are more present than ever before. There are multiple game events with huge prizes every year. Especially young people get in touch with video games through their smartphone, tablet, console or computer early in their lives. Of course that is mainly because of the fact that with today's technology it is possible to not only make the games more realistic but also more creative and therefore increase the fun a player has playing the game. Serious games which are often used to improve certain skills could be used to increase the motivation and learning success of pupils, especially in science courses like physics or mathematics where many students struggle to find the value of gaining the knowledge in this domain. Furthermore pupils could find it easier to understand and learn certain aspects by applying them in games. This is amongst other things one of the goals of the *UFrameIT* project.

*Contribution* In this paper I want to present the main idea of the *UFrameIT* project, the different categories of serious games and their respective areas of application shown on examples. It is also important to exhibit the various approaches every section of serious games has as well as their advantages and disadvantages in terms of learning efficiency. Furthermore I will explain the main properties of the *FrameIT* method which is used in the *UFrameIT* project, the current status and future goals of it.

*Related Work* Similar approaches like the knowledge based *FrameIT* method are hard to find. Nevertheless, a few are worth mentioning. The **ELPI** framework for example is quite similar to the **MMT** knowledge management system which the *UFrameIT* project members use in their *FrameIT* approach. While both use one language for syntax and rules writing the main difference is that *ELPI* focuses on the performance and *MMT* on the easy usability of the system. [RM19]

Another project which has a similar goal as *UFrameIT* is **Specware**. This knowledge based software development system helps the user generate provably correct code. [Ins21]

A last project I want to mention in this section is the **PhET interactive simulations concept**. These mini-games developed by members of the University of Colorado focus on helping students or anyone who is interested to improve certain skills like learning fractions by cutting cake pieces. [Col21]

*Overview* In the next section I will explain the main idea of the *UFrameIT* framework using the current mini-game **FrameWorld** and its differences to other serious games. In section 3 I will describe the underlying *FrameIT* method more in detail followed by the current status and future goals. I will finish this paper with a conclusion.

## 2 The UFrameIT Framework

The first idea of the *FrameIT* method came up in a paper from 2012 written at the Jacob's University in Bremen. Denis Rochau then implemented a first version of it as a bachelor thesis based on the Unreal Engine. After a few years some students revived the former project in cooperation with the chair of visual computing at the Friedrich-Alexander University in Erlangen in 2019. The current result is the *UFrameIT* project which is lead and supervised by professor Michael Kohlhase.

The main idea of the *UFrameIT* project is to create a framework which can help developers to focus on game interactions while the underlying knowledge is managed by the knowledge management system *MMT*. Furthermore the project members want to implement their own games for pupils. The current example of such a game is called the *FrameWorld* mini-game which is free to play on every platform. [Tea21b]

### 2.1 The FrameWorld Mini-Game

In this mini-game the player can learn trigonometric relations by logically calculating a tree's height. The player starts in first person perspective in a forest-like environment with gadgets which are shown in the bottom of the screen and can be switched between each other. The **pointer gadget** and **line gadget** are used to set points and measure distances on the ground and with the help of

the **protractor gadget** the player is able to measure angles. After generating these points, distances and angles are saved and called **facts**. The task of the game is to apply them to the *OppositeLenScroll* to compute the correct height of the tree.

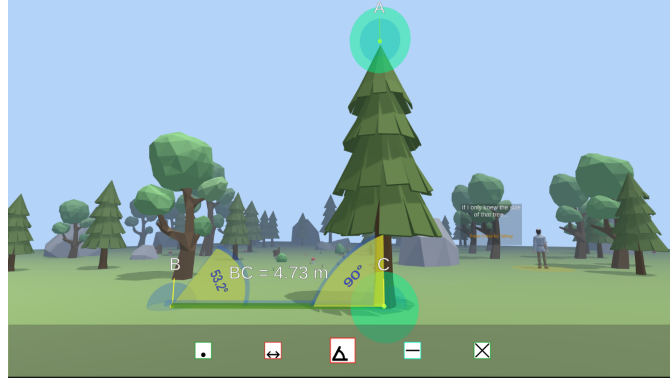


Figure 1: Scene from *FrameWorld* mini-game

After generating some *facts* they are shown in the game menu as named, quadratic game items. Along with that *FrameWorld* displays the **Opposite-LenScroll** consisting of a description of how the opposite length of a triangle can be calculated and slots for the corresponding *facts* (see Figure 2 picture 1). The player can then grab the *fact* items and pull them into the slots which represent every variable in the *OppositeLenScroll* formula. Additional to that the game offers a hint button for every slot highlighting the correct *fact* for it. After the player put a *fact* item in every slot the "Apply" button can be pressed to check whether the assignment was correct. If there was a mistake rain will appear to show that the player must try again. In the case of success the game ends with a firework rewarding the player with the height of the tree.

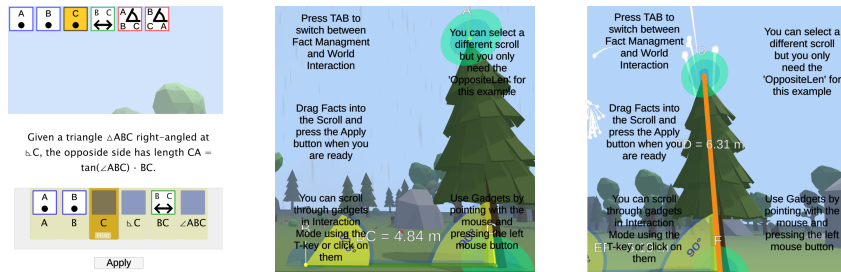


Figure 2: From left to right: Requesting a Hint, Rain in the event of failure, Fireworks in the event of success

## 2.2 Serious Games and Similar Approaches

As professor Micheal Zyda from the University of Southern California describes it, 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] Therefore there are multiple areas of application for serious games which can be divided into four main categories. [I'S]

*Serious Games* The games in this category are those which focus on learning or improving certain skills without asking for the reasons. One good example for that would be **America's Army**. This first person shooter game is quite successful, free to play and used by the US Army to recruit and train new members. For example if a soldier has some problems with the rifle range or completing obstacle courses some sergeants let them train these actions virtually in the game. After a few hours of training on the computer big improvements can be seen in real life according to a staff sergeant from Fort Benning. [Zyd05] Another game which fits into this category and is quite similar to *FrameWorld* is called **Dragonbox Elements**. In this game young players can learn geometry by finding certain geometric forms out of a set of lines and circles. Like *FrameWorld* *Dragonbox Elements* tries to make it easier for pupils to learn mathematical aspects. [Püt21]



Figure 3: Screenshots from *America's Army* and *Dragonbox Elements*

*Serious Educational Games* A perfect example for serious educational games is *FrameWorld*. The main difference to conventional serious games is that they not only want to train skills but focus on the gain of learning new aspects. The key idea behind this is to make it clear to the player why it is important to learn or improve specific skills. In *FrameWorld* the player finds out that learning trigonometric relations can be important in real life, for example for calculating heights of objects only by measuring angles and distances on the ground.

*Simulations* Games in this section are characterized by the fact that they simulate dangerous scenarios from which to learn. In principle they have the same function as serious games or serious educational games but offer no score or



economic value to the player. While games of the previous sections normally give feedback to the player in one form or another, simulations do not. One of the most successful simulator games is **Microsoft Flight Simulator** where the player can learn how to fly in a real life environment with the help of google maps. But also other games like **Pulse!!** fit perfectly into this category. *Pulse!!* is used for simulating surgeries to train new surgeons. The main advantage is that "you can kill as many people as you need in virtual reality in order to harm no one in real life" [Dob21], as Claudia Johnston, one of the developers of the game aptly describes it.



Figure 4: Scene from *Microsoft Flight Simulator*

*Virtual Worlds* The last main category of serious games are virtual worlds. One popular ambassador of this section would be **Minecraft** which is one of the most successful games in history. Especially its educational version is nowadays even played in schools in the US already. While most of the games from the other categories are played offline a key advantage for virtual worlds is the communication aspect the player gets by interacting with multiple other players around the globe. Because the idea of *Minecraft* is to build worlds as the player likes them to be creativity can be improved a lot. That is why there are countless possibilities to apply *Minecraft*. In some schools in the US for example the pupils learn chemistry by creating chemical experiments in their own world and visiting those created by other classmates or players around the world. Another example would be visiting worlds where historical buildings are recreated in a realistic way. In fact, the focus is on the width of the worlds created by countless players. [Tea21a]



Figure 5: Screenshots from *Minecraft* [Tea21a]

### 2.3 Problems of Creating Serious Games

Although all these approaches are quite nice for learning new things there are of course a few problems developers of serious games have to face.

First of all it is fundamental to think about the learning objective and what exactly the player should learn by playing the game. Sometimes it can be quite difficult to find the right balance between quality and quantity of learning objects.

Another aspect a developer has to take into account is the domain simulation which means basically how the game responds to the player. There are three approaches for this. Ignoring the players actions, punishing failure or rewarding success. As I mentioned earlier simulations normally ignore the player's actions because they want to simulate the environment as realistically as possible. All other kinds of serious games, however, need to react to success or failure to motivate the players and evaluate their behaviour. This can be done via a credit system which is often used for example in role-playing games to progress in the story line and unlock more levels or quests. In contrast, many *jump 'n' run* games prefer "punishing" the player by resetting his progress and force him to restart a mission.

The next problem would be to think about how to create fun playing the game. This aspect is one of the most difficult as it requires a lot of domain knowledge in psychology and didactics.

It is also important handling problems and progression. This basically means thinking about the level design and in which way the players should learn. In previous examples I presented some approaches preferring a more realistic level design and using it to show the players how exactly they could apply the learned aspects in real life like it is handled in *FrameWorld*, *America's Army* or *Microsoft Flight Simulator*. In contrast, *Dragonbox Elements* for example prefers a much simpler and more abstract level design to make it more interesting for young players.

The last main aspect a serious game developer has to consider is which game mechanics and deployment should be used. [ILM11] Maybe a "retro" approach as a conventional computer game would attract older players as it reminds them of their own gaming history. For children a modern, visually fascinating smart-phone app would be ideal to arouse interest.

In summary, it can be said that the main difficulty for serious games creators is to prioritize the categories mentioned as well as to deal with psychology and didactics.

### 3 The FrameIT Method

The basic concept behind the *FrameIT* method is to divide labor between a game engine and a mathematical knowledge management system what I will explain more in detail in the following. *UFrameIT* is an implementation of this method and can be divided into three parts: the game engine **Unity**, which is used for the game mechanics, the *MMT* knowledge management system for managing the underlying knowledge and a back end server providing communication between *Unity* and *MMT*. [Koh+20]

#### 3.1 Unity Game Engine

*Unity* is a state of the art, multi-platform game engine which offers several advantages for a usage of it for implementing the *FrameIT* method. As an industry standard offering a game engine for several popular games like *Fallguys* it built a huge community in the past few years offering many tutorials as well as free materials and assets. Furthermore it is available for basically every platform including virtual and adaptive reality which are the future of gaming platforms. In addition to that *Unity* offers an interface for communicating with *RESTful APIs*, which is necessary for the interaction with the *MMT* system. [Koh+20]



Figure 6: Scene from *Fallguys*, developed with the help of *Unity*

### 3.2 The MMT system

The main idea of the *MMT* knowledge management system is to provide a general foundation and logic-independent framework for creating formal systems. In *UFrameIT* it enables validity checking in form of giving hints as well as computing the results. *MMT* has three major functions used for *UFrameIT*. [Koh+20]

*Storing* *MMT* divides input knowledge into theories which are lists of typed constant declarations formalized by  $c: E [=e] [\# N]$ .  $c$  acts as an identifier while  $E$  is a well typed expression.  $e$  stands for the type and definiens and  $N$  for some notation. Therefore it is possible to represent a large scale of formal knowledge consisting of function type, predicate symbols, axioms inference rules or theorems. In addition to that it is also possible for *theories* to import knowledge from other *theories*. [Koh+20]

*Relating* A **view**  $v: S \rightsquigarrow T$  relates two *theories*  $S$  and  $T$ , mapping every declaration in  $S$  to a  $T$ -expression. This means for example instantiating abstract theorems like trigonometric identities. In *UFrameIT* this is useful for example for computing a tree's height only by knowing the projected width and the enclosed angle. [Koh+20]

*Combining* To combine the knowledge *MMT* generates **pushouts** in the category of *theories* and *views*. These *pushouts* are used for example for translating abstract conclusions into the context of a concrete situation. This means basically framing the solution formula as the abstract calculation. [Koh+20]

*Example* For a better understanding of the method, I would like to explain it in more detail using the tree example mentioned in section 2.1. Initially the game provides the player with the problem of calculating a tree's height along with some background knowledge. This knowledge is represented as *facts* and *scrolls*. *Facts* represent mainly the knowledge gained by the player in the form of measured distances and angles as well as labeled 3D points set by the player, such as  $A = (1,0,0)$ . *Scrolls* on the other hand, are much like mathematical theorems. The basic idea of them is to supplement the concept of *facts* with a mechanism in order to generate new *facts*. Every *scroll* has a list of required *facts* in order to use it. In our example the player has access to the *Opposite-LenScroll* which requires three point *facts*  $a, b, c$ , one  $90^\circ$  angle  $\angle abc$ , as well as the enclosed angle of the triangle  $\angle cab$  and the distance  $|ab|$  to compute the opposite length of  $bc$  via the tangent function (see Figure 7). [Koh+20]

## OppositeLen Scroll

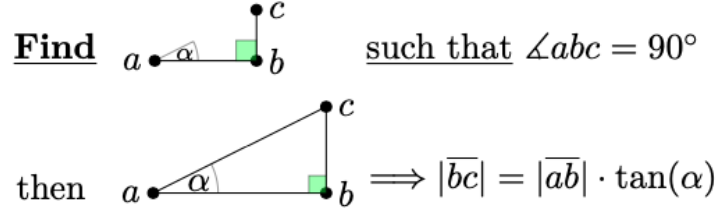


Figure 7: *OppositeLenScroll* from *FrameWorld* [Koh+20]

We can divide our problem into two sections in the form of the virtual world side on the one hand and the theory and knowledge management side on the other as it is shown in Figure 9. In the latter we can represent the underlying formula as a problem/solution pair where the problem part contains the required variables and the solution the actual equation to finally compute the solution using the required variables. We assume that the player generated some *facts* using the integrated *line*, *pointer* and *protractor gadget*. In our case these *facts* would be three points  $E$ ,  $F$  and  $G$  where  $E$  represents a random point on the ground and  $F$  and  $G$  the bottom and top of the tree the player wants to know the height of. Furthermore the player measured a distance  $|\overline{EF}|$  as well as the two angles  $\angle EFG = 90^\circ$  and  $\angle GEF = 45^\circ$  meeting all of the requirements of the *OppositeLenScroll* (see Figure 9). This gained knowledge in the form of collected *facts* is then immediately communicated to the *MMT* system and stored in a situation part on the knowledge side.

By pulling the *fact* items into the *scroll* variable slots as described in section 2.1 *MMT* creates an **application view** implementing the required variables with the associated and generated values. Through these *views* it is also possible to show hints to the players if they did something wrong and highlight the right *facts* which belong to the corresponding variable slot in the *scroll*. Furthermore

```
view:
A->E, B->F, C->G
distance(A,B) -> distance(E,F)
angleAt(B) -> angleAt(F)
angleAt(A) -> angleAt(E)
```

Figure 8: Generated *view* in *MMT* [Koh+20]

more *MMT* knows if a requirement is not met, for example if  $\angle EFG \neq 90^\circ$ , and can exactly tell where the error is. In the final step a *pushout* is computed by the *MMT* system in the form of applying the values generated by the *view* into the solution formula and the result is finally computed. In addition to that *MMT* ensures simplifying terms and reporting the success of the player to the game engine. [Koh+20]

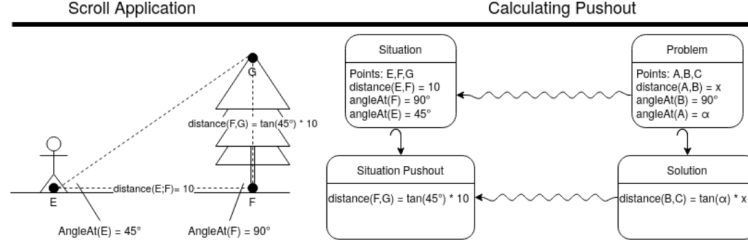


Figure 9: *FrameIT* process model [Koh+20]

### 3.3 Back End Server

For the communication between *Unity* and *MMT* during the game a back end server is needed. It provides adding *facts*, generating *views*, requesting *pushout* computations and listing available *scrolls* by a *RESTful-interface* which transmits the corresponding payloads via *JSON* data format. These *JSON* files can then be interpreted and finally be handled by *Unity*. The communication is triggered by three main events.

The first one would be **Game World Triggers** which automatically send requests during the interaction with the game world. However, in this simple example these were not used.

Another trigger is the **Fact List Modification**. Every time the *fact* list is changed by the player generating new *facts*, it is reported to the server. This happens by sending the *fact* details via a *HTTP* request. *MMT* then checks this information for validity, saves it, upon success, and lastly sends the generated declaration identifier back to *Unity*.

The last event which triggers the communication is the **Attempt of Scroll Application**. When the variable slots of a *scroll* are tried to be filled the information of it is sent to the server and packaged into a *view* by *MMT*. This *view* will then be type checked and its result reported back to *Unity*. If successful the game engine then requests the computation *pushout*. [Koh+20]

## 4 Current Status and Future Goals

In the last couple of years the *UFrameIT* team provided an own website for the project [Tea21b] where news, demo videos and all important information can be found. At the moment the members work on multiple issues. A demo version available for the platforms MacOS, Windows and Linux are finished and can be downloaded for free. Furthermore the design of more "*FrameWorlds*" has already begun as well as the implementation of a toolkit helping future game developers. This toolkit should make it easy to create problems in the game given the *scroll* which should be used to solve it. In addition to that the team regularly updates their advertising videos to expand their reach to get more people involved in the project.

Future goals would be to optimize the compatibility of *Android* tablets as they are the most used tablets in schools as well as implementing plugins for game engines. Moreover the team wants to attract more students with innovative research topics, and of course implement more complex games, some of which can even be developed by teachers with no or little coding experience.

## 5 Evaluation

The knowledge based approach the *UFrameIT* project team members used to implement their framework of course has some advantages and disadvantages. While *ELPI*, the most similar approach to *MMT*, concentrates on the speed, an argument against using *MMT* would be that because *MMT* focuses on the usability of prototyping for example in the form of providing flexible notations and error messages its performance suffers from explicitly dispatching to its rules. That is why there could be some problems in handling larger amounts of input. [RM19] Furthermore the explicit modelling of the background knowledge leads to having to consider edge cases which could be worked around in code-the-behaviour approaches.

In spite of the mentioned arguments it makes a lot of sense to use *MMT* as the knowledge management system for the *FrameIT* method. First of all by separating the development workflow into a game and a knowledge side as described in the example in section 3.2 it reduces the probability of mistakes being made between domain experts and developers. In addition to that reusing the knowledge becomes possible as the background knowledge is independent of the implementation itself and therefore the formalization process only has to be done once. The last main advantage would be the aspect of reusing game design. That is basically because of the game can be updated by adapting the underlying *theory* graph which is also described more in detail in the example of section 3.2. [Koh+20]

## 6 Conclusion

In summary there are quite a few problems for the development of serious games like combining the game development itself with didactics and domain knowledge. The *UFrameIT* project, however, offers a good approach of handling these problems with its use of background knowledge management. This aspect makes it easier for teachers to create their own levels and helps them to motivate their students. The pupils also get a huge advantage from this by learning complex scientific concepts while having fun. So all in all in the near future when the framework is fully elaborated offering multiple worlds including many applicable *scrolls* the *UFrameIT* project implementing the *FrameIT* method has a huge potential to improve didactics in schools.



## References

- [Col21] University of Colorado. *PhET Interactive Simulations*. visited on 03/14/2021. URL: <https://phet.colorado.edu>.
- [Dob21] Jason Dobson. *Comment on Pulse!!* visited on 03/14/2021. URL: [https://www.gamasutra.com/view/news/102373/SGS\\_Feature\\_Pulse\\_First\\_Person\\_Healthcare\\_System\\_Simulation.php](https://www.gamasutra.com/view/news/102373/SGS_Feature_Pulse_First_Person_Healthcare_System_Simulation.php).
- [ILM11] B. C. Ibanez, J. Labat, and B. Marne. *Conceptual and Technical Frameworks for Serious Games*. 2011. URL: [https://www.researchgate.net/publication/271202681\\_Conceptual\\_and\\_Technical\\_Frameworks\\_for\\_Serious\\_Games](https://www.researchgate.net/publication/271202681_Conceptual_and_Technical_Frameworks_for_Serious_Games).
- [Ins21] Kestrel Institute. *The Specware System*. visited on 03/14/2021. URL: <https://www.kestrel.edu/research/specware/index.html>.
- [Koh+20] M. Kohlhase et al. *FrameIT: Detangling Knowledge Management from Game Design in Serious Games*. 2020. URL: <https://kwarc.info/people/mkohlhase/papers/cicm20-frameit.pdf>.
- [Püt21] Sarah Pützer. *Comment on Dragonbox Elements*. visited on 03/14/2021. URL: <https://www.spielbar.de/node/146982#bild146255>.
- [RM19] F. Rabe and D. Müller. *Rapid Prototyping Formal Systems in MMT: 5 Case Studies*. 2019. URL: [https://www.researchgate.net/publication/336796817\\_Rapid\\_Prototyping\\_Formal\\_Systems\\_in\\_MMT\\_5\\_Case\\_Studies](https://www.researchgate.net/publication/336796817_Rapid_Prototyping_Formal_Systems_in_MMT_5_Case_Studies).
- [Tea21a] Minecraft Developer Team. *Minecraft: Education Edition*. visited on 03/14/2021. URL: <https://education.minecraft.net>.
- [Tea21b] The UFrameIT Team. *The UFrameIT Project*. visited on 03/14/2021. URL: <https://uframeit.org>.
- [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](https://doi.org/10.1109/MC.2005.297).