Schriftenreihe Fachbereich Informatik

2017-1

Informatik
Hauptcampus

HOCHSCHULE TRIER

Gamification Under the Hood
Using Game Technology for Building an Interactive Math Learning System

Jan Simon and Georg J. Schneider
Department of Computer Sciences, University of Applied Sciences Trier, Schneidershof, 54293 Trier, Germany
{jan.simon, georg.schneider}@hochschule-trier.de

Keywords: e-Learning Hardware and Software, Cross-platform Development, Game Engines, Children's Education using Computer Support and K12 Students, Mobile Learning (M-learning).

Abstract: The papers describes the implementation of an interactive math learning system using the game engine Unity3D. We will focus on the implementation aspects of interactive learning systems using this game engine, since the game development approach as well as the use of this specific game engine bring along several benefits, like device independence and reactivity.

1 INTRODUCTION

Interactive math learning systems have a great benefit for K12 students, since they offer one more way for the pupils to interact and experiment with complex aspects of the math curriculum. They can explore the nature of formulas, the relation to the function graph and the underlying meaning on their own. In (Escuder and Furner 2011) the positive impact of the interactive math system "Geogebra" has been illustrated. However, although if the system is widely used in schools, this system does not offer an immediate feedback in a way that functions graphs can be dragged or stretched, while the parameters in the formula are updated immediately and vice versa. Therefore, we suggest having an immediate feedback in both directions, in a way that a student can immediately follow the changes and see how and how strong the impacts are caused by the manipulation (Blanke, Schneider 2011). In order to guarantee an almost immediate feedback, traditional software development methods using an event-based process (Potts et al. 2014) are well suited for the classical UI development. On the contrary, the software development process for games using a game loop is targeted to offer immediate feedback.

An additional challenge is imposed by the very heterogeneous hardware environment in schools, i.e. different operating systems and different output media, like PC's, tablets or interactive white boards. A recent development, which has already a strong impact in the enterprises (French, Guo, Shim et. al. 2014) makes the situation even more complex: Students bring their own devices and want to use the device to support the learning process in the classroom as well as at home (Song 2014). Therefore, systems that are targeted to support learning in school environments should be able to adapt to different devices and screen resolutions. Similar results have been obtained from a user study, we have conducted with pupils form a local high school (Schneider, Ubl 2016). However, systems that ran on student's hardware lower the costs for specialized hardware for the schools, which makes them even more attractive for both, schools and pupils who can take the system at home.

In this paper, we will describe a general approach to develop device independent, highly interactive learning systems in a flexible an efficient way. We will illustrate our approach in more detail by describing the implementation of our interactive math application.

2 SOFTWARE ARCHITECTURE AND IMPLEMENTATION

In the following, we will illustrate the manifold constraints, which have to regarded in order to realize a promising application for a school environment.

Afterwards we will describe the most fundamental aspects of our implementation using the

game engine Unity3D (Unity 2016), which nicely deals with these constraints.

2.1 Technical Requirements for Educational Applications

One of the core challenges of developing educational applications is the wide range of hardware that can realistically be used when wanting to learn or teach with the program in the classrooms, the private homes of teachers and students as well as "on the go"'. Any educational application that forces schools or students to switch from one device with their preferred operating system, e.g. a desktop or laptop computer, to a different device with a different operating system causes a huge obstacle and will most certainly not be used voluntarily. The current trend BYOD (bring your own device) has also arrived at schools. Teachers and students expect that applications run both, on full-scale computers as well as tablets and other mobile devices. Therefore, desktop and mobile versions of the software have to be provided. Especially for mobile applications this issue is amplified. With Android and iOS sharing the market in almost equal parts, supporting both operating systems is crucial, as no group of teachers or class of students is willing to change to a different device only for the sake of a math application. As an additional challenge operating systems with a smaller market share, such as the Linux operating systems and Windows phones must be taken into consideration as well. These are certainly less common but anyone using any of these systems must either be outfitted by the school with a new device or left out of the learning group. Both options are not acceptable.

On top of this, the list of challenges continues. The devices that run the application can have virtually any kind of screen resolution and aspect ratio. A "hardwired" interface for each resolution and aspect ratio option is both, not appealing and unfeasible. Consequently, the application needs to adapt the user interface in an intelligent and flexible way, without elements overlapping each other or getting cut off and without wasting valuable screen "real estate".

Users also interact with the application in different ways on the different devices: they use traditional mouse and keyboard when working at a desktop computer, touchpad and keyboard on laptops and notebooks, and touchscreen input for tablets, phones and other touchscreen-enabled devices. Developing for and testing these input methods in a seamless and hassle-free way is vital.
To summarize the points mentioned above we can state that an educational application will only be highly accepted from schools if it is a multi-platform application providing a flexible user interface that nicely adapts to the screen resolution and aspect ratio and offers various methods of input, depending on the device.

For this reason, we have chosen Unity3D, which covers these requirements and has hence been selected as the foundation for the reimplementation of the Suremath application (Schneider, Ubl 2016). Using the Mono architecture, Unity supports an extremely wide range of platforms, e.g. Windows, OSX, Linux, HTML with WebGL, Android, iOS, Windows Phone and Blackberry. It is possible to write an application once and to deploy it to any target platform. There is no need for maintaining different branches of the software or code branches that "customize" the behaviour of the program based on the target system.

2.2 Screen-independent User Interface

A number of requirements have to be fulfilled for a graphical user interface to be truly screen-independent, that is, flexible in regards to both resolution and aspect ratio.

The position of each interface element has to be defined in relative terms as opposed to absolute pixel locations for example. Unity3D uses an anchoring system, where elements can be anchored to regions and either fill them up completely or stick to one edge of the anchor. The concept also supports padding (distances between elements) which helps making the interface look more appealing and it improves the readability. This way, elements are dynamically positioned, stick to that position and are independent of the screen.

However 2D graphics cannot simply be scaled arbitrarily. An image for a button background that is 256x64 pixels large, for example, will only look acceptable if it is scaled in a certain interval. If the resolution or aspect ratio forces the element to change its scale dramatically, the difference between the asset graphic and on-screen element resolution will force the renderer to up- or downscale the graphic, generally resulting in sub-optimal image quality (blurriness, aliasing, missing or double pixel rows or columns).

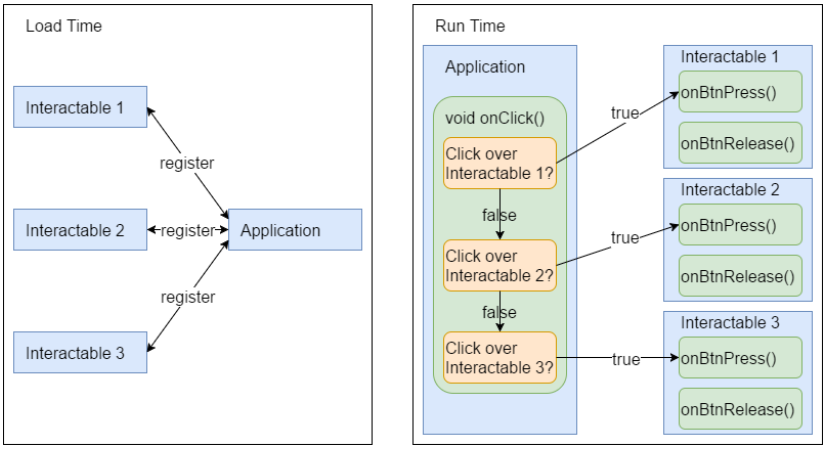

Figure 1: Traditional Approach.

This problem is addressed within the game engine with a so-called "smart scaling feature". As the engine calls 2D graphics, sprites can be broken down into nine separate regions: the four corners, the four edges and the centre. Hence, when a graphic is scaled, it will not simply stretch the original image, but instead create a new graphic with the corners and edges in the correct positions. Only the middle region has to fill up the centre to reach the necessary size. Neither the corners, the edges or the middle are scaled in this process. The first two are at their original size and the latter is simply repeated (i.e. wrapped) as necessary. This allows the application to define graphics for arbitrarily sized elements (buttons, windows, sliders etc.) that will look acceptable at any scale.

2.3 Consistent User Input

In a school environment, it is very likely that users will run the application on multiple devices at the same time and not just exclusively on one system. For example, a teacher could prepare an exercise for the students at home on her desktop computer with mouse and keyboard. The following day she will teach the class at school using a touchscreen-enabled surface table. Another situation might be that students have just worked on an in-application assignment at school in the computer room. Now they are in the bus, finishing the assignment or experimenting with the application.

In order to support these use-cases, it is important to implement a consistent input system as opposed to different and individual interaction methods that depend on whether the users clicks using the mouse or taps on the screen with a finger.

This design philosophy prohibits certain gestures for providing a consistent interaction with the application over the different platforms: there exists no right-clicking with finger tapping, and multi-touch finger inputs cannot be emulated with a single mouse pointer for example. This does not imply that right-clicking and multi-touch input are obsolete. The challenge is that intuitive solutions must be derived and implemented that counterbalance the shortcomings of the various input devices in order to provide a consistent user experience throughout. For examples a long tap could have the same effect as a right-click when using a touchscreen-enabled device; scrolling the mouse wheel can zoom in and out when a mouse is being used, replacing the multi-touch finger pinch gesture.

2.4 Event-based versus Game Loop Approaches

The traditional method of writing programs with user interfaces is to use an event-based software architecture. Figure 1 visualizes the approach.

During load time of the application, all interactables (Buttons, Textboxes, Sliders etc.) either register themselves to or are registered by the application. When the application detects user input during run time, it iterates through the list of registered interactables, checks whether or not the interactable should react to the input and, if so, calls a respective callback or handler function, which can then execute any actions that need to happen as a response to the user input.

The framework can then poll for user input, determine which registered interactable, if any, are affected, and signal the user input to the element. For the latter, a number of call-back functions are usually written (using function pointers in C/C++ for

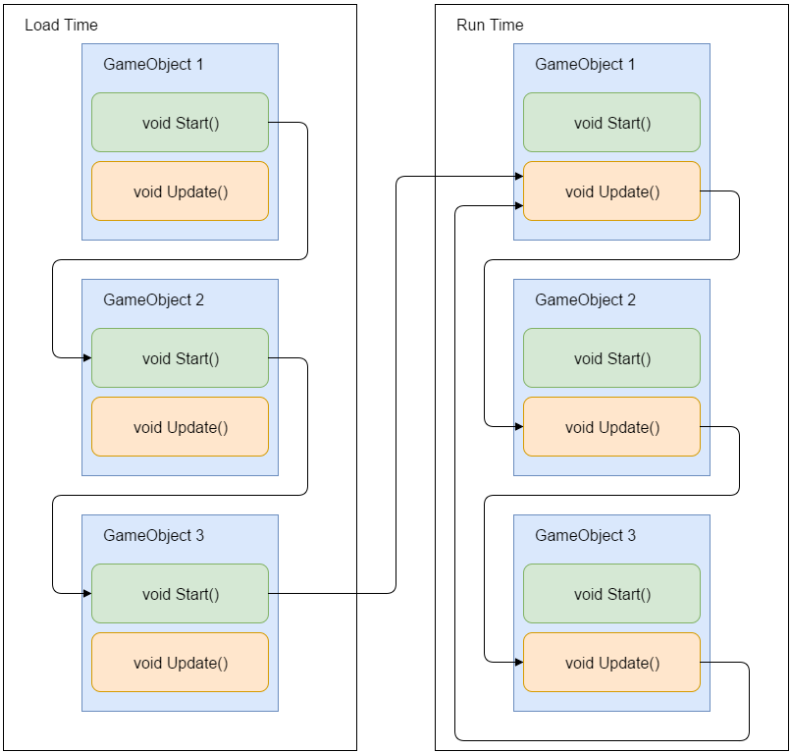

Figure 2: Game-based Approach.

example, or abstract base classes or interfaces and polymorphism in Java).

To illustrate the approach: a set of functions `onButtonPress()` and `onButtonRelease()` for example are implemented. At load time, the button registers itself as an interactable. The user clicks on the button, the framework polls for input, detects the click, iterates through the list of registered interactables, determines that the position of the mouse cursor overlaps with the bounding box of the button and calls the `onButtonPress()` function that the button implements.

Using Unity in place of a more common interface-orientated framework means that a game loop approach will be used instead. Figure 2 visualizes the way that Unity operates.

Elements do not use an event system to determine when they need to react, change or re-draw themselves, instead a "polling" approach is used. During load time of the application, Unity iterates through all Game Objects and calls the respective `Start()` methods, if they exist. This allows elements to initialize data if necessary. Once all `Start()` methods have been called, the game loop starts. Elements are now iterated continuously and `Update()` methods are called, giving elements the opportunity to change their behaviour dynamically during run time. Rendering is done automatically by the Unity engine, after the `Update()` method has been called and garbage collection is also handled automatically, so no explicit method to release allocated memory exists either. The `Update()` method of each element (or "Game Object" as it is called within Unity) is executed once every frame. This approach is common in games (and was hence adopted by Unity) because the scene that is being rendered changes constantly. The traditional event-based system is more geared towards static less reactive programs, where none or very small areas of the screen or window change.

2.5 Implementation

The Suremath application is set up as a 2D project within Unity3D and consists of one scene only. A scene in Unity is a collection of cameras, lights and renderables (such as 3D models or primitives). When Unity3D is used in a more conventional setting to develop a video game, a scene is often equivalent to a level or stage. This way only the necessary data for the current location needs to be loaded and kept in the RAM, minimizing the memory footprint. Suremath is not split up into

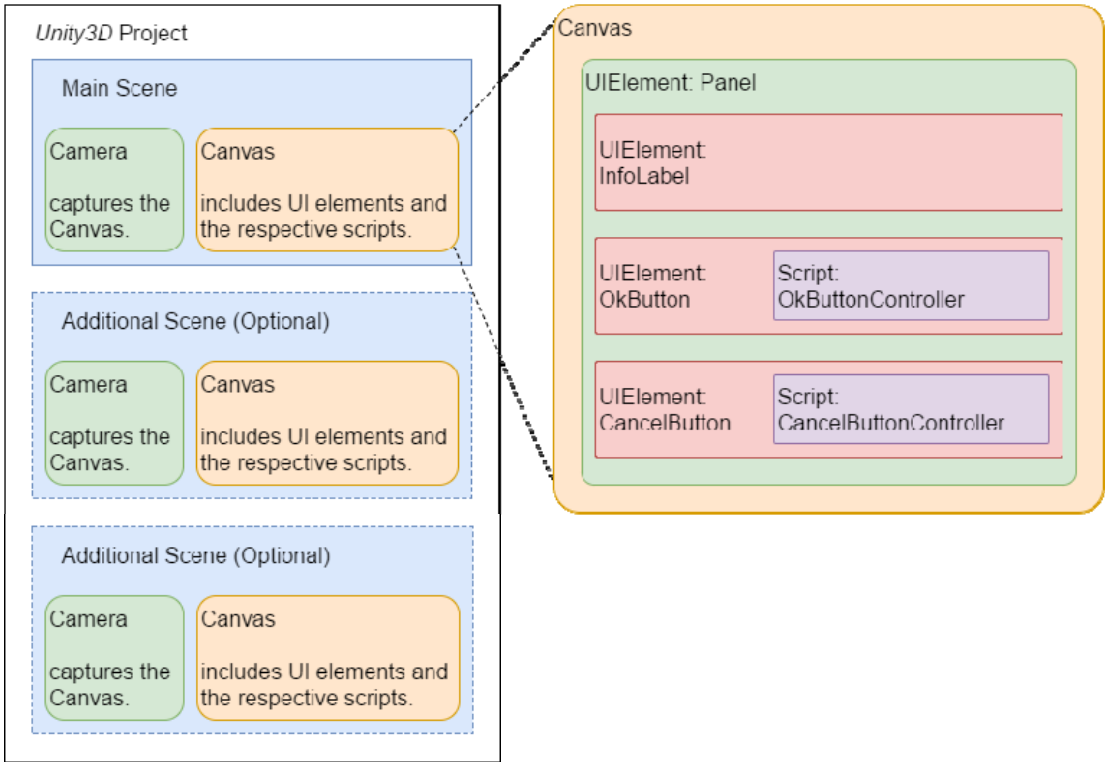

Figure 3: Diagram of a Unity3D project.

different scenes or levels. For very complex applications, however, which require multiple screens or modes of usage, it is always possible to branch the project out into any number of scenes, which also reduces the amount of required memory - since unnecessary resources can be unloaded - and makes it possible to encapsulate functionality that is only relevant to one screen into that respective scene.

Since Suremath does not require 3D rendering, animation, lighting or any other advanced rendering features, its single scene is set up in a very simple fashion. A canvas that spans the entire screens serves as the background and every element (label, button, panel and so on) is parented to this canvas. The only other object in the scene hierarchy is a very basic camera that captures the canvas and renders it to the screen. All scripts that control the behaviour of the application are assigned to the respective elements directly. A button that displays a message would e.g. contain a script that triggers the message to appear. This direct assignment of scripts to components (interface elements in this case) is in accordance with Unity3D's underlying philosophy and helps to keep even more complicated interfaces neatly arranged. There is no need for large script files that control a multitude of elements, instead each element has a usually fairly concise script that controls exclusively that element.

Figure 3 shows the diagram of a Unity3D project, featuring one main scene. Additional scenes can be added if needed, in which case each scene can hold the data of a screen in the application. Each scene has its own camera and canvas, which holds all UI elements (panels, labels, buttons, lists and so on) for the respective screen and, if needed, a controller script that defines the behaviour for the element (i.e. what happens when a button is clicked or a list item is selected).

Figure 3 also gives an overview about single- and multiple-scene projects in Unity3D and serves to illustrate how UI elements are arranged in a canvas, with scripts attached directly to them.

2.6 Preliminary Results

We are developing Suremath in cooperation with students from a local high school: a group of roughly ten 12th year students that have shown interest in the subject of eLearning. We periodically let the students test the application and collect feedback. We then evaluate and prioritize the feedback, implement the necessary changes and schedule another test session, which guarantees a very

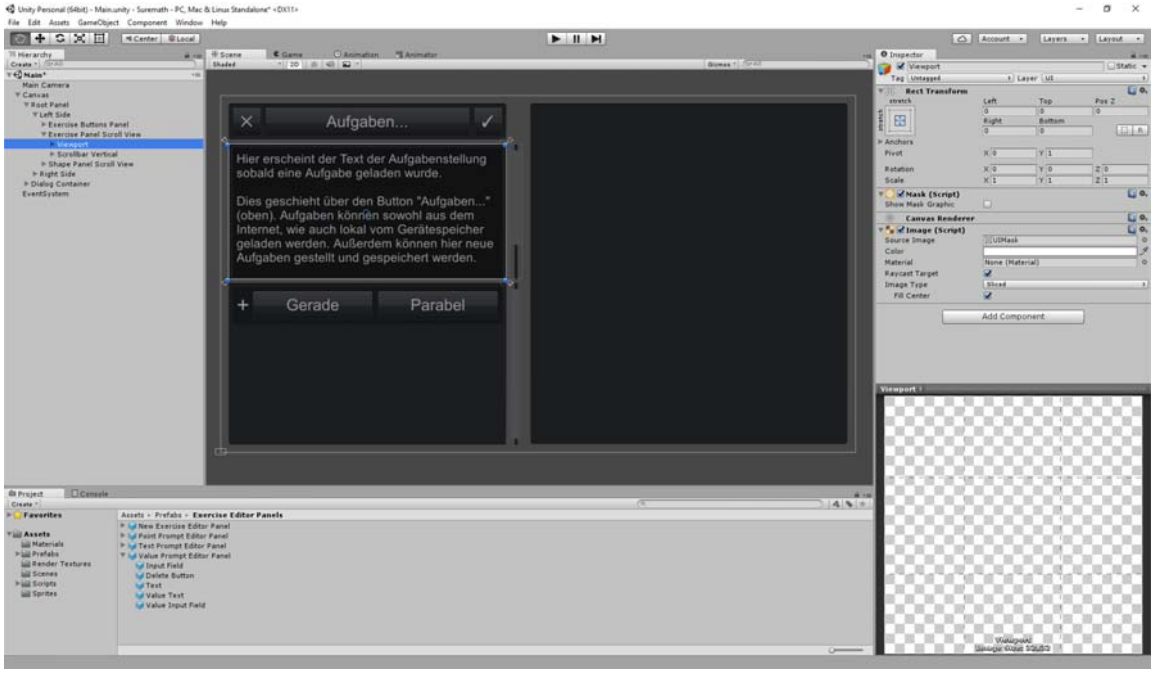

Figure 4: Unity3D development environment.

dynamic, agile and end-user-orientated development cycle.

We have found that students are very quick to point out any inconsistencies they discover during the usage of the software. They have consistently discovered bugs or glitches. We have taught them to write down reproducible steps to force the errors and motivated them to come up with hypotheses for the reasons. This procedure is valuable for us to be able to quickly eliminate any bugs and has been received very well by the students, who tend to make a contest out of it. On the other hand the students become familiar with a structured procedure, for software testing.

Together with the students we also discuss future features and prioritize them for a road map for the continued development. Since they are very close to their fellow students, which are confronted with the learning material, the students are an extremely precious resource.

3 CONCLUSIONS

In this paper, we have presented a game-based approach to build highly interactive learning systems. Changing the concept from a traditional point, click and draw pattern to a game loop, which draws each element in each frame makes the application much more reactive.

Supporting different output devices is a crucial point in software development generally. In the case of a school environment, where learners bring their own devices and they expect that the learning applications behave like their favourite applications, the situation is even more demanding when creating software for this target group. Hence the use of the Unity3D engine proved to be a successful approach because of the possibility to write platform independent applications. Therefore it is not only possible to offer the application on various platforms but also to reduce the amount of work to support different platforms since the source code remains the same (see Figure 4).

The system is currently used from a group of advanced pupils from a local high school, which are creating a concept for a workshop with younger pupils and teachers with the use of our system.

We are planning to offer an Android version of our software in the Play store in the near future, as soon as we have finished our work with the online authoring system.

Right now we still support only parabolas. In the future we want to support more shapes, preferably arbitrary polynomial functions. We plan to integrate a scripting system, which allows users to create complex exercises. A further feature is the possibility to script camera movements. We want to display text messages and play audio files as well. Furthermore we intend to display, remove and

animate arbitrary shapes. This would allow for virtual lessons that do not just ask students to enter calculations and results like a simple exercise but additionally explain new concepts through these visual means.

ACKNOWLEDGEMENTS

We especially want to express our gratitude to the Auguste-Viktoria-Gymnasium, Trier, Germany especially Miss Anne Kress and her pupils who supported our work by participating in our workshops and still continue working with us on this topic.

REFERENCES

Blanke, D., Schneider, G., 2011, TOM A multi-touch System for learning math, in *Proceedings CSEDU 2011, 6. - 9. 5. 2011*, Noordwijkerhout, The Netherlands.

Escuder, A., Furner, J. M., 2011, The Impact of Geogebra in Math Teacher's Professional Development, in *International Conference on Technologies in Collegiate Mathematics*, edited by P. Bogacki et al. (Department of Mathematics and Statistics Old Dominion University, Norfolk, USA, 2011), pp. 76–84.

French, A. M., Guo C., Shim, J.P. 2014, Current Status, Issues, and Future of Bring Your Own Device (BYOD*), Communications of the Association for Information Systems: Vol. 35, Article 10.*

Ng, W., 2015, Mobile Learning: BYOD and Personalised *Learning, in: New Digital Technology in Education, Springer International Publishing, 2015, pp. 171-189.*

Potts, J., Hildebrandt, N., Gordon, J., Castillo, C., 2014, JavaFX, Getting Started with JavaFX, Oracle Corporation, *Release 8, E50607-02.*

Schneider,G., Ubl, I., 2016, Suremath, User study and related (re-)implementation of a multitouch application for learning math, *in Proceedings: CSEDU 2016 - 8th International Conference on Computer Supported Education, Rome, Italy, 21. - 23. 4. 2016.*

Song, Y., 2014, Bring Your Own Device (BYOD) for seamless science inquiry in a primary school, Computers & Education, Elsevier, *Volume 74, May 2014, Pages 50-60, ISSN 0360-1315.*

Unity, 2016, https://unity3d.com download 19.12.2016.