Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Master's Thesis

General Architecture for Development of Multiplatform E-Learning Applications

Bc. Štěpán Tesař

Supervisor: Ing. Filip Ježek

Study Program: Biomedical engineering and informatics

Field of Study: Biomedical informatics

May 11, 2015

Acknowledgements

I would like to express an unending gratitude to my parents for all the support during my time at the university, and to my supervisor, Ing. Filip Ježek, for guidance and advice with this work. Furthermore, I would like to thank the faculty staff for the provided challenges and opportunities, and my colleagues and fellow students for warm memories and mutual support during the studies.

Authors declaration

I hereby declare that I have completed this thesis independently and that I have listed all the information sources with accordance to the Methodical instructions of compliance with the ethical principles for writing university theses.

Prohlášení autora práce

Prohlašuji, že jsem práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 11. Května 2015 ....

Abstrakt

V rámci této práce byl vytvořen konceptuální pracovní postup a implementovány otevřené nástroje, které umožňují rapidní vývoj výukového softwaru. Cílem bylo vytvoření standardizovaného prostředí, do kterého je nutné dodat pouze specifický obsah aplikace, přičemž veškeré uživatelské rozhraní a aplikační logika je již zajištěna. Pro tyto účely bylo nutné propojit několik existujících projektů a programovacích nástrojů. Dále bylo třeba vytvořit silně modulární uživatelské prostředí, zcela nezávislé na obsahu.

Práce se zaměřuje na výukové aplikace, využitelné v univerzitním prostředí. Aplikace vytvořené pomocí navrženého postupu tak musí splňovat základní požadavky na výukové aplikace, a tedy nikoliv pouze zobrazovat obsah, ale zároveň simulovat výuku. Studentům jsou tedy prezentovány otázky a testy, a aplikace se zachová adekvátně dle dané odpovědi. Z pokročilejších cílů práce je možné jmenovat například umožnění vizualizace fyziologických modelů, nebo snadné nasazení výsledných aplikací na různé aktuálně populární zařízení používané studenty, jako jsou tablety, notebooku a stolní počítače s různými operačními systémy.

Abstract

In this master's work, a conceptual workflow and open tools were implemented, to provide an instrument for rapid development of educational software. The aim was to create a standardized environment, where only the case specific content has to be provided, and all of the user interface and functional behavior is handled by the supporting software. For these purposes, several existing projects and frameworks had to be interconnected, and a strongly content-independent, modular user interface had to be constructed.

The focus was on educational applications applicable in a university environment. Therefore, the finished application has to fulfil basic expectations of an e-learning software, meaning that it does not only display the content, but also provides a simulated classroom behaviour. The students can be presented with tasks and quizzes they have to solve, and the application reacts differently according to their answers. There are also some more advanced targets, such as necessity to display various visualizations of physiological models, or possibility to deploy the finished application on current technological solutions used by students, such as tablets, notebooks and desktop computers with various operation systems.

Contents

Abbreviations

- 2D/3D - 2 or 3 dimensional
- API - Application Programming Interface
- CSS - Cascade Style Sheets[1]
- DPI - Dots per Inch - a pixel density measurement
- GUI - Graphical User Interface
- HTML - HyperText Markup language[1]
- MVC - Model, View, Controller (design pattern)
- MVVM - Model, View, ViewModel (design pattern)
- SVG - Scalable Vector Graphics (graphics file format)
- UI - User Interface

[1] A W3C language standard (see www.w3schools.com)

Glossary

Graph structure - "In mathematics, and more specifically in graph theory, a graph is a representation of a set of objects where some pairs of objects are connected by links." [1]

Portrait and Landscape mode - are two possible orientations of a screen of a mobile device, that has one edge of the screen longer than the perpendicular one (i.e. the screen ratio is not 1:1). The 'portrait' mode refers to the state when the screen is bigger along the vertical axis, making it narrow, but high. This mode is typical for smart phones, and it is more suitable for reading or taking portrait pictures (hence the name). The 'landscape' mode is the state when the width of the screen exceeds its height, making it more suitable for watching movies or browsing pictures and, as the name suggests, taking pictures of landscapes.

Responsive design - refers to a design of user interface, that responds to the change of the screen size, not only in the terms of scaling the GUI components (see Scalability), but also by rearranging them in such a way, which allows easy reading and navigation through the screen content, and matches conventions typical for devices with such screen sizes. E.g. on small mobile devices are usually the menus hidden, and shown after a press of a button. On larger screens, the menus can be shown all the time along one side of the screen, since the main content would not take up all of the width anyway.

Scalability - a feature of graphical user interface (GUI), which allows non-fixed screen size. The GUI will adapt as the screen size changes by scaling the GUI components as well (at least along one of the screen axes). Scalability can be implemented for example by creating a 'Responsive Design' (see previous term).

[1] cited from en.wikipedia.org/wiki/Graph_(mathematics)

Chapter 1

Introduction

Developing any application requires not only a specific skills, but also takes a large amount of time. There are tools, frameworks and principles to reduce this time, that target wide market and usually provide vast number of tools that the user-programmer can use to create his application. A user interface (UI) usually has to be created from a scratch by a designer, and programmers have to set up the background logic, data storage and resolve deployment issues etc. What is missing in this approach is an area-specific focus, that removes some of the options and freedom in customization of the application, but largely reduces the time required to get from concept to final deployed application. "A well-defined e-Learning architecture helps dramatically in cutting costs and increasing the speed of development while it sustains quality results."[12] This is greatly possible in educational applications, which generally have the same, or at least very similar, requirements and goals.

"The tutorial software creation process is slowly becoming a blend and a combination of pedagogical experiences and the creativity of enthusiasts. It is mostly work for specialized teams using highly specialized development tools and it is beginning to look more and more like an engineering project."[16] This thesis covers a concept that tries to allow even less technically grammar people to create simple educational tools, and to reduce the application development process into only two steps. Firstly, creating the content itself - writing texts and providing relevant pictures or videos, which, necessarily, has to be done by the specific field expert (E.g. medical teachers - doctors). And secondly, setting this content up in an already existing environment, with a limited or zero programming knowledge, which could possibly be also done by the same person who collected or created the content, thus reducing time and cost. The first step is always necessary, and is the only part that is truly unique to any educational application. Focus of this work is then to provide such an environment, where required types of content could be created or imported, interconnected and presented.

The aim is to create a relatively simple instrument that provides the same options as a simple office presentation (text, pictures, videos), but also adds value to this concept by providing a graph-like structure of content (as opposed to a purely linear structure of a presentation), which by itself provides many options of use (thumbnails of pictures in text, quizzes, interactive content in general etc.), and also by providing options to simply create both desktop and mobile applications. Because of focus on technical and medical education, the support to implement a mathematical model in the Modelica language[1] of physical systems is also provided. This is also done in such a way, which does not restrain the programmer from implementing advanced behavior. Also, it creates a unified interface to speed up the connection of the model to the application. As much as the connection to the rest of the application can be made easy, the design and creation of the model itself will remain a task for a person with particular skills. "On the level of constructing simulation models, cooperation has to be established among system engineers (skilled in mathematical modeling), physiologists and physicians eventually, if the model is supposed to be applied in clinical medicine."[14]

[1] 'language to conveniently model complex physical systems' - www.modelica.org

Chapter 2

Main goals

In this chapter, major objectives of this thesis are set, along with a general path, which should lead to their accomplishment. Several problems were encountered during the development, and while the general objectives remained the same, some revisions of priorities had to be made multiple times.

2.1 Objective

Creating an application is an extremely time-consuming task including several layers of focus. There are many methods of approach to software development[1], but they ultimately all include similar steps. From collecting requirements, through designing the application structure and visual style, selecting the target platform and required testing. Relative amount of time spent on different tasks vary from method to method, but it is a matter of fact, that by using the correct tools, the time can be reduced. For eLearning applications, a majority of the development process is similar: the actors (students, teachers), the general requirements, and even more specific functional demands are usually virtually the same (see 2.2).

The demand for eLearning applications is indisputable[17]. There are several popular tools for rapid eLearning application development[5], which are very general and provides lot of tools, that are usually unnecessary for highly specific applications in a specific field (such as medical training). For example, if there is no need to extensively customise the user interface, a huge part of development process suddenly ceases to exist. Keeping the same UI throughout multiple (eLearning) applications can also be beneficial for the users, since they will not need to familiarize with new the environment in each application.

As a review by Justin Ferriman states, "the best program for eLearning development really comes down to personal preference as well as the type of content you are training to"[5]. This reasoning led to a concept of a very new and specific tool for eLearning development. One that would even more reduce the time required for development by implementing as much of common properties of the final applications, and thus removing the need to develop them for each application separately. This is possible for a price of a less customizable visual, but even functional, components of the application. Brief research of current software solutions has also shown, that complex mathematical model simulation (with visualizations and for mobile devices) frameworks are basically non-existent. Which is a further motivation for creating such platform.

2.2 Application requirements

The created product will serve as a tool to create educational applications. As opposed to entertainment or business applications, such software has a purpose of providing useful information in an engaging way. It does not generally serve to store or organize data, to simplify any work process, tasks or communication. The usual audience (students) should be able to receive necessary context or visual aid. To define specific requirements, it must be firstly defined what creates a good eLearning architecture. For this specification, provided is an enumeration of "characteristics of a good e-Learning architecture" by Ray Jimenez[12]:

1. It identifies the quality outcomes suitable to e-Learners
2. The components and its purposes are specific
3. It is easy to replicate and transfer from one person to another
4. It is documented, shared, and distributed
5. The software selected fits a specific function
6. It is easy to tweak and/or improve
7. It is easy to determine the cost and to manage it
8. It aids in meeting timelines
9. The skill sets required are easy to determine and assess[1]

Further in this section, there is a definition of what the finished product should provide. This set is derived from the previous list of good characteristics, and it directly dictates how the framework for creating such applications has to be conceived. There are both functional and non-functional requirements, which are divided to logical categories:

1. Content types
   - Text
   - Images
   - Videos
   - Interactive animations
2. Content organization
   - Separate screens

[1] The whole list is a citation[12]

    - Combined content
    - Information transfer
    - Graph structure

3. User interaction

    - Switching between screens
    - Manipulating screen
    - Unified interface

2.2.1 Content types

As mentioned in previous paragraphs, the truly unique part of any educational application is its basic content. Every computer presentation already provides most of the types of content, that are necessary for most educational purposes.

Text is obviously the most important, basic type of content that can be provided. The interface should allow basic formatting options, so that the produced text is customizable and pleasant to read.

Images are a basic visual aid component of most presentations. Therefore it is not necessary to uphold its importance in any educational project.

Videos similarly as images can serve multiple purposes, from simple looping animations to underline principle that is being explained, to longer informative films.

Interactive animations are one of the goals of this work, that can elevate the applications above simple presentation. By giving the student the ability to manipulate with the content, comes a far easier way of understanding the principles behind the presented concepts. In this thesis the goal goes even further, to provide an actual simulation of a physical system, based on modelled physical rules written in the Modellica language, where the user can influence its inputs, and observe how does the output change.

2.2.2 Content organization

In order to make an application useful, it is important not only to provide relevant content of certain quality, but also to find a way to present the content as a whole. Each unit of content should provide certain information, but it also has to fit the overall picture.

Separate screens of content should stand independent of each other, and user should be able to differentiate between them without any problem, so there is no loss of context or confusion. A "screen" is meant to be a unit of content, not physically constrained - as in computer screen - but grouped by specific topic or purpose. Since the application should provide all the necessary material for the given subject, it has to be divided into well-arranged units. Such units should then be presented separately, but obviously allow transitions from and to each other, as described further.

Combined content is simply a way of expressing the categories (described in section 2.2.1) are not fixed, and the media can be mixed together. It is very usual to include a small informative graphic in a long text, or vice versa - to provide short explanation under a large image.

Information transfer renders important, if the application wants to present a content depending on user's previous actions. This simple mechanism allows to create a 'cross-roads' content, where the student has to make a decision. For example to select a correct answer. Based on this answer, the other content can be manipulated, or even a completely different content can be displayed.

Graph structure is an implication of the previous paragraph, but also provides useful options by itself. For example, there can be a smaller preview of a larger image embedded in a long text, and after clicking this image, the user can be directed to a screen with the original, large image, with further note. The original approach was to create a tree structure, but it is much more convenient to allow for loops in the graph, since even for a complex quiz structures, there is usually some common content, and the little differences based on previous answers can be solved by the information transfer principle.

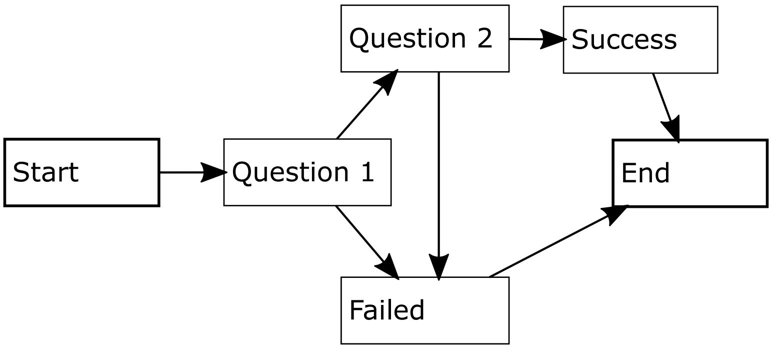


Figure 2.1: Example of a possible content structure

2.2.3 User interaction

The way user interacts with the content is partially implied by the structure of the content itself. Presented here are additional principles that should be abode in order to further help the user to orient in the given software.

Switching between screens should be as simple as in any other presentation, in both directions. As the content will be connected in an unconstrained graph structure, it is important to correctly manage the history of displaying content, which would enable a suitable navigation system for the user.

Manipulating screen must not disturb the user and has to be simple and intuitive. Manipulating the screen means navigating within one content unit. E.g scrolling a text or a picture that does not fit the computer resolution. In case of interactive content, the interface providing the user's input should have clear function and be always visible when desired.

Unified interface is a simple way to make the user familiar with all parts of the application very quickly. After the student uses one control mechanism of the application, this mechanism should be accessible in the same way and position, regardless of in what state the application is in (in terms of displayed content). The simplest example is a "close window" button in most of current desktop operating systems, which is always in the top right corner of the window.

2.3 Production requirements

A set of non-functional requirements for the software development and deployment is summarised here. The developed framework should fulfil most of these requirements without a need to further modify its output.

- Platform independence
- Scalability
- Permissive licence
- Low financial cost
- Interconnection with existing frameworks

Platform independence is the strongest production requirement that has been set for this work. In today's world of personal mobile devices, the need to publish an application on a broadest spectrum of them is obvious[8]. It ensures that the final application will be deployable on most of the current devices, but also produces the need to carefully select the development strategy.

Scalability only expands the previous requirement. Since the development is to be kept platform independent, there is no guarantee of a medium on which the application will run. The graphical user interface (GUI) has to be designed with this problem in mind, and therefore be automatically or at least very easily adjustable to any potential screen size.

Permissive licence makes the developed framework widely usable, since it is far simpler to promote a tool that can be further modified, improved, but most importantly, a tool that the user can use without any commitment. This approach also creates a community that can add to the software, and the software contributes to a global library of free projects that anyone can utilize. An interesting quote: "Consider the saying 'If you want something done right, do it yourself.' in the case of product and service development, the adage holds true for users in part because a manufacturer cannot know what users want as well as they themselves do."[23], also points out more advantages of an open source approach. When users are allowed to adapt the product to maximally fit their needs, the potential reach becomes practically limitless.

Low financial cost is required mainly because of the academic background of the production and also to make the previous goal possible. This obviously creates certain challenges in finding a suitable tools for development. Staying true to this and the previous goal will produce a framework that can create applications that might help not only a classic university students, but also provide knowledge to areas where is a lack of classic education. "In recent years, the development of eLearning had resulted in less emphasis on openness and access, and more emphasis on commercialization, profit or at least cost-recovery in post-secondary education. Nevertheless, distance education still has a critical role in serving those who have difficulties for personal, social or economic reasons in accessing conventional campus-based education. (...) Distance education, whether based on e-learning or older technologies of print and broadcasting, remains a powerful tool for economic development and personal advancement"[22]

Interconnection with existing frameworks is specifically required to support the functional Interactive Animations requirement. A framework exposing a convenient simulation interface (Bodylight framework [11]) has been already in development in the faculty department, and it is most convenient to cooperate with its authors to expand it and utilize it for the goals of this work. This will be required to allow simulation capabilities in the framework.

Chapter 3

Analysis and design proposal

Present here, is an accumulated knowledge from the researched area, which has been party done prior to, but by part also during, the main implementation process. There are several layers of implementation, and each presents its own problems. Each layer then produced new findings, that had to be retrospectively applied to the previous ones, which often led to changes of design. This chapter describes design of each layer by its own, but also in the context of the overall work.

3.1 Tools selection

Essential part of a general architecture development is the selection of suitable tools. Since the aim is to allow its usage to as many developers as possible, a sufficiently widespread technologies had to be selected, while keeping the main goals of work in mind.

3.1.1 Physiological model

There are limited tools for a physiological system modelling. Recently, the discussion in this area has shown large benefits of Modelica in the field of physiological modelling[7]. Also covered by Kofranek, Matejak and Privitzer: "Modelica allows a much clearer than other simulation environments, to express the physiological nature of the modeled reality."[15].

The one logical alternative, simulink[1], is less appropriate as well for such application. "The structure of the model in Simulink corresponds to the structure of computational steps, while the Structure of Modelica model reflects the structure of the modeled physiological reality.[15].

There are implementations of model oriented design in applications (For example in efficient building design, see [19]), but these are one-instance products without published open framework and tools, that could be reused for rapid application development in other fields. A quick look into the current market shows that there is only one framework in development that aims to provide rapid development of applications with Modelica models, and that is Bodylight. This framework is by far not a competition, since a large part of it

[1] 'a block diagram environment for multidomain simulation and Model-Based Design.' - www.mathworks.com/products/simulink/

will be used within this work, which will help to further expand the original framework in the sense of creating a new layer of UI, that is both independent on Bodylight, but is also prepared to be connected with it. Furthermore, a 3D support is not a strong domain of the original Bodylight framework, which is a feature that the new framework should provide with ease.

As previously mentioned in 2.3, bodylight is a framework developed by fellow department members, and is designed to allow controlled simulation of a model created in the Modelica language. This framework also includes procedures for creating the visualisation of such simulation. In this work, it will be necessary to extract the basic parts, and create matching interface for the newly created solution of visualisation.

Bodylight is developed in the .NET framework[2], and written in C# and F# programming languages. The end-user interface is implemented in XAML and distributed via Microsoft Silverlight platform (adapted from [11]). It might be challenging to create a package fully compatible with another tool. It was therefore important to select such programs that will have the same code basis and provide sufficient binding support.

3.1.2 Graphical engine

3.1.2.1 Options

Since one of the targets is to provide environment, where users will be able to integrate 2D and 3D content, it is required to select such development environment, that will contain a graphical engine with such capabilities. There are further requirements for the selected graphical engine: There have to be tools, developed specifically for such engine, that will allow simple, visual composition of the application content. The tools also have to be customisable, so that a UI composition and settings system, fitting the needs of the future framework, can be created. During development (and partly even when using the framework for creating more complex applications), there will be some programming work involved. This also has to be integrated seamlessly with the graphical engine.

There are several graphical engines, that are popular among developers of 3D games, and the question of selecting the right one is what everyone has to face before starting the development process. One comparison summarises the presented choices as such: "It's ultimately up to you to decide which one works best for your project. One thing is certain; there is not a shortage of game engines at your disposal."[18] while reviewing currently most promising platforms, which is Unity, Source 2, Unreal Engine 4 and CryEngine. Since the goal of this thesis is not creating a 3D game, but a system for developing elearning applications, which will generally not require a particularly strong graphical engine (only basic 3D animations without advanced effects are needed). Thus, the conclusion of the previously mentioned review: "All of these game engines would be a great choice for your game development process. Unity is great for mobile, 2D and 3D games. Unreal Engine 4 gives you the ability to create games with photorealistic graphics or simple 2D side scrollers with a reasonable pricing model of a 5% royalty, and CryENGINE has amazing graphical capabilities as well, and next-gen platform features with a pricing model that can be more

[2] 'a technology that supports building and running the next generation of applications and XML Web services' - msdn.microsoft.com

appealing than UE4's depending on your studio"[18]. This gives a first hint, that Unity might be the engine of choice for this thesis purposes.

Another big requirement is a possibility of development for multiple platforms (at least major mobile - iOS, Android, Windows - and desktop - Windows, Mac OSX, Linux). The environment also has to be financially affordable (preferably free). Research in this area gave a different set of results (Xamarin, Kivy, Unity, Titanium Appcelerator, Sencha Touch, Enyo) than the graphical engines in previous paragraph, with only one item common to both sets: Unity. As mentioned in a related Jarcas Studios editorial[21], Unity has several advantages over other multi-platform environments. As the author states: "The dev process has been painless. The community is great. The Asset Store is a lifesaver. Load time is only a second or two. Package size is large, but not ridiculous"[21]. Further research only confirmed, that Unity is the perfect tool for this thesis assignment. Another reviewer concludes: "It's very easy to get started with Unity, and you can instantly see the result of what you are working on in the editor without having to wait for the game to compile and build. This is huge! From the interface all the way down to the workflow and how art is imported, you can see the brilliant execution the Unity's idea of 'democratizing game development' in what we think is the best game software around today."[25], after listing the 10 reasons why to choose Unity (list adapted from the previously cited article[25]):

1. Free to get started with
2. Multi-platform
3. Thriving, supportive community
4. Asset store
5. Scripting languages (Javascript, C#)
6. Ability to create 2D games
7. Multiplayer support
8. Online tutorials and classes
9. Unity conferences
10. Ease of use

After researching other possibilities, none matched the needs of this thesis as closely as Unity.

3.1.2.2 Unity

Unity[3] is an extensive software designed for development of games. It is currently a well performing, yet still expanding platform, that is rising in popularity among developers. It

[3] 'a flexible and powerful development platform for creating multiplatform 3D and 2D games and interactive experiences' - unity3d.com/unity

has been selected for the application logic and UI development for several reasons. Although some of the presented arguments do not apply to this thesis requirements (multiplayer support or conferences), it provides several key features:

- Multi-platform development support
- .NET framework utilisation
- Free license available
- Integrated tools for UI development
- Extensive 3D support

Multi-platformity is a strong part of Unity, since it is based on the Mono Project [4]. The official Unity web page states that 'Unity now supports 21(!) platforms'[20]. The supported platforms include all major operating systems for both desktop and mobile devices, as well as several gaming consoles and smart TVs. There is also option to deploy the application for web player, which works through plugin for the majority of current internet browsers. Furthermore, Unity developers recently released a working preview of deployment to WebGL. "WebGL is a 3d graphics library built into the browser, which allows JavaScript programs to do 3d rendering inside any supported browser without requiring any plug-ins."[4], which ensures lasting support for web deployment in the future.

The Mono Project is an open-source .NET framework implementation, which guarantees compatibility with other projects created within the same framework. This is important for interconnection with the Bodylight framework, which also utilizes .NET. This should ensure correct operation of Bodylight libraries within Unity, with only little alterations to the framework code.

Unity provides a free license in the shape of Unity 5 Personal Edition, which includes most of the features, but lacks advanced developer support. There are some advantages of the professional (paid) versions of Unity, which are discussed in section 4.2.11. Still, the most important features are available in the free version as well. Most interesting are components for effortless composition of GUI. These components can be also used for more elaborate scenes, where they do not serve to create UI, but as parts of an animation. This is all backed by a strong 3D engine, which allows elaborate 3D scenes, object interaction, and is perfect for the purpose of creating model visualizations.

Unity was also decided upon to explore a new branch of Bodylight possibilities. The original visual part of Bodylight is implemented in Silverlight, and further aim of the framework is to create a meta-structure that could be easily deployed on any architecture, reimplementing the visual side in Unity will provide valuable insight into different approach, which can then be used when creating the meta-specifications. It was also not clear, if this connection will be even possible, and the experience of the struggle will be very valuable for future development of bodylight alone.

[4] 'an open source implementation of Microsoft's .NET Framework' - www.mono-project.com

3.1.3 Formated text containing pictures

To allow simple content creation, there has to be a specified way of describing the text structure, including position of pictures. "With the advent of the World Wide Web, structured text (in the form of HTML) has become a dominant medium for online information"[3]. It is convenient to use the most widespread markup language: HTML.

The Hyper-Text Markup Language is, in its basic extend, a very simple tool for formatting text. Although modern language specifications try to separate the visual formatting from this language, and keep only the structural purpose[10], it is still one of the simplest, and easily understandable options for text formatting. "It was designed to be sufficiently simple so as to be easily produced by both people and programs (...)".[2] It is a perfect compromise between extend and simplicity, for importing text into the Unity environment. Since the thesis does not strive to provide online connectivity from applications created by the framework, the task of providing simple html parsing support has a reasonable extend, and would allow the users to provide large-scale texts with embedded pictures and easily specified formatting. Using html also provides infinite options of reuse of the content without any need of additional editing. As Saarela and Wium Lee state: "... the semantics of the various elements are well known: all browsers and search engines know that the "H1" element indicates a first-level headline. Thus, HTML has achieved a unique position as a device-independent, ubiquitous document format."[24]. It is fair to mention, that the cited article has been published in the year 1994. Since then, html has only developed even more, and several new technologies have been built upon it. The classic tags and usage are therefore almost a basic knowledge of any person grammar in technology-related fields, and as such is extremely easy to learn even for complete amateurs.

In Unity there already is a native support for a very tiny subset of HTML formatting in its 'rich-text component', which allows usage of basic tags to specify text color and bold or italic style. The community discussion on this matter also clearly provides a wider demand of integrating full html format support natively in Unity, which would probably also open options of CSS formatting.

3.2 Architecture design

The selected tools have to be interconnected to form a distributable architecture, which can then be used by programmers to create individual applications.

3.2.1 Connecting Bodylight framework

It is necessary to create such structure, which will allow creation of simple applications without models, and therefore without the need to embed Bodylight framework with them, while keeping a full support of all other features. On the other hand, it is equally important, that the framework can be connected instantly on demand, and that the programmer only has to implement the case specific behavior.

Bodylight implements an MVVM design pattern (Model, View, ViewModel), as visualised in figure 3.1. There are structures that allow simple data-binding with visual components of Silverlight. One of the challenges of this thesis is to use this interface to connect an entirely different 'View' part, matching the framework's interface and using the available structures.

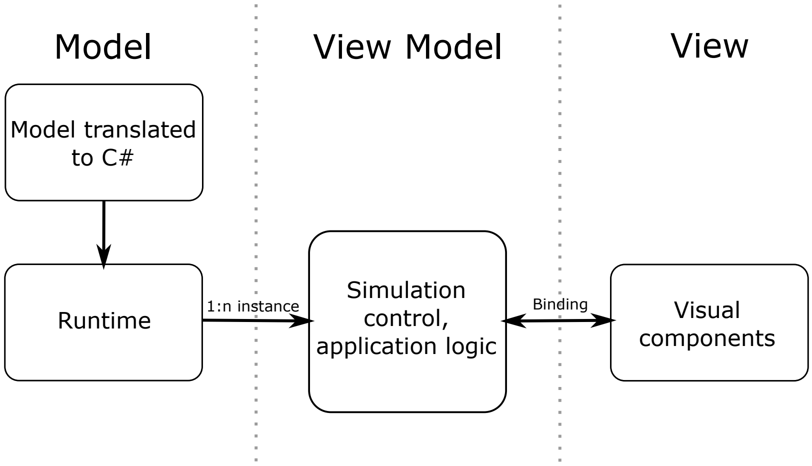


Figure 3.1: Bodylight structure, adapted from MEDSOFT 2013[11]

Since Unity is an extensive environment with its own graphical engine, there are some principles in place that will dictate how the interconnection can be implemented. For example, Unity has a built-in frame-rate system, that sends a signal to all components to refresh themselves. This feature can prove useful, but will also mean, that the standard MVVM approach will be impossible, since every visual component has to have an attached class, implementing given Unity interface, that takes control of the visual side of the component. This means that the final structure will be more inclined to an MVC pattern (Model, View, Controller), as seen in figure 3.2. By connecting the Bodylight ViewModel variables through an Event-listener, or Observer patterns, the binding-like nature of the connection can be maintained.

This approach also solves one major problem before it even occurs. The Bodylight simulation has a real time synchronization mechanism, which allows production of simulations independent on processor load. This means, that the simulation has to be provided by a 'tick', that will determine the next derivation step in the simulation. Because of the Unity frame rate system, the visual components can be simply synchronized with the simulation by invoking the tick after Unity has finished rendering each frame. This will cause the simulation to keep the time dependent derivations in synchronization with actual time, and the users will not experience speed variations. Just to clarify, this will not prevent stuttering, or other types of lag caused by heavy load of computer components. But, on the contrary, will prevent these lags to slow down the simulation output.

3.2.2 Application structure

Apart from connecting Bodylight, the structure of application in Unity has to be carefully designed as well. The final framework has to be well structured, so if any programmer wants to do greater alterations, there are no big obstacles in his way. At the same time though, the framework is primarily designed to provide visual tools for application composition, and

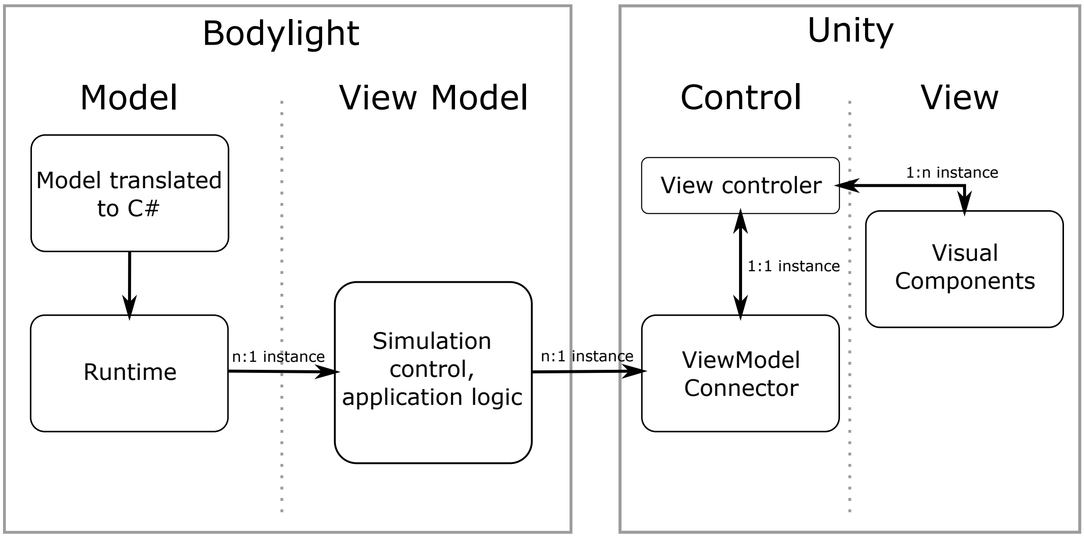


Figure 3.2: Bodylight connected to Unity

this will require a more complex structure of mutually dependent parts, that can handle minimal user input, and compose the final application.

Content structure as a graph, is one of the main goals. As opposed to a linear content structure, in the content graph, any content can be linked to arbitrary number of other available contents, as described in 2.2.2. To allow this, a controller overseeing all the single units of content has to be created. This class will manage signals from the displayed content, and eventually display another content.

As seen in the figure 3.3, this component will have an array of previous components. This will allow easy 'back' function, without the necessity of keeping the whole virtual content graph. Each content node holds references to its target nodes, and only the path back to the starting node is being preserved. This is the optimal approach, because at no point of the application is it necessary to know all links in the graph.

It is also useful to introduce an optional 'state' parameter, which can be used to slightly alter the content. As seen in the diagram, this parameter is passed to the method, that handles displaying of new content, and it will be passed to the new content via its Show method. This parameter can be used when making only slight changes to the content. The user can implement a simple method that will switch between states, and can for example set slightly different parameters to the Model, or display a different image, in otherwise identical content.

The Show and Hide methods of Content can be used to run routines when initializing, or disposing content that needs special treatment (such as custom models). This custom behaviour has to be implemented by the user.

The diagram also shows, that each content has a 'next' property, which points to another instance of the Content type. By introducing this property, simple linear structure can take place, in cases where there is no desire to split it into multiple paths. Otherwise, if there is such desire, the Content also holds reference to the ContentController, and can call its ShowContent method with any parameters, since the list of possible contents will be available to the user when creating the application. These assignments will be done visually through Unity GUI, making most users completely separated from the need to write any code.

The abstract marking of the Content component illustrates, that there are multiple types of content, and each can act differently. Multiple implementations will be provided to the user, and he can then only add the specific texts or visuals. There should be at least two basic types of Content: html and model. Html will serve to display text and pictures, while model will display a custom visualisation of a Modelica model. This will allow automatic alteration of the Unity GUI, so user is presented with relevant options, and can visually manipulate the content instances.

The realization in unity is not as straightforward, as it might seem from the previous diagram, but the principle is shown correctly.

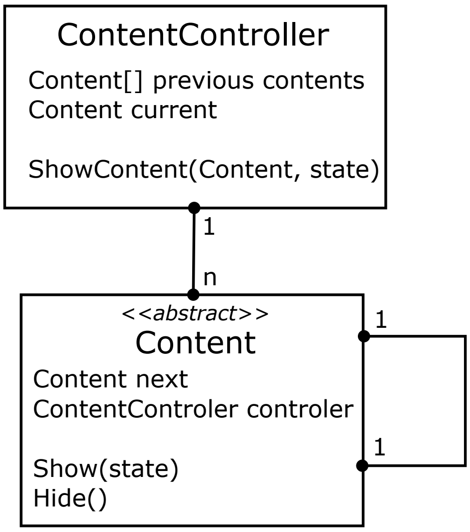


Figure 3.3: Diagram of class structure taking care of content organisation

Model interface structure is one of the more complex features, and is an example of a part that had to be changed several times in order to resolve newly discovered problems. The resulting solution is pictured in the following figure.

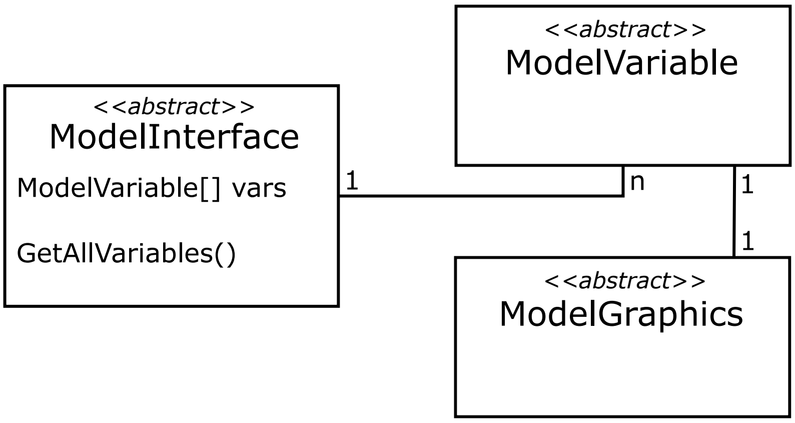


Figure 3.4: General structure of the classes used to connect a Modelica model

All the classes in this diagram are labelled abstract, since all of them have multiple specific implementations. The ModelVariable encapsulates behaviour of a single variable used in the model. It is necessary to use an extra class for each variable, and not only a list of primitive types, to provide automatic behaviour. Apart from a value, the class also contains a specification of interface for the visual component setting. Once created, the user can manipulate its value freely, but the framework core will automatically take care of its visualisation.

The ModelGraphics represents the graphical component that is used to either visualise, or to manipulate the variable. Several different variable types, with matching visual components have to be implemented to cover majority of possible use-cases.

ModelInterface implementation has to be left onto the user, since it is completely model-specific. In the figure 3.2, this class is represented by the 'ViewModel connector' block. This is where user has to specify create instances for the model variables, assign observers to the model and handle its changing. The visual components will provide new values, or interface for displaying them. This is one of a few parts where user actually has to write code, but by providing suitable implementations of ModelVariables, it should be made as simple as possible. One of the requirements of this thesis, is an example of model backed application, which can serve as a template for future developers.

3.3 GUI design

With knowing the main goals, and the required content structure, as described in the section 3.2.2, it is easy to define basic components. The GUI has to be universal enough, so that it can support varying demands of developers, but still simple enough and user friendly.

The first question that comes into mind when thinking about multiple-content application, is the navigation. From the diagram of content controller and content (3.3), it is obvious that there have to be controls to move back in history, and also to go forward to the next content in the linear structure. In case of a disambiguation, it is necessary to display more buttons, that will redirect the user to different specified contents. This basic facts were already obvious in the very early stages of development, and even the first prototypes contained them (see picture 3.5).

3.3.1 Layout

After some prototype testing, and early stage application development, several principles were set. It was decided that the main navigation bar, as seen in the aforementioned picture, should remain the same throughout the application, with only a simple message, that can, for example, shortly describe the current connect. Extended functionality (disambiguation buttons) should be in a separate place, which will be displayed above the static navigation bar. In the diagram on figure 3.6 below, this are is labelled 'Extended command bar'. Furthermore, the inputs and outputs of model should also be unified, and be displayed around the screen sides, which should provide space for multiple controls. All components should only be displayed when needed, and hidden if empty, so they do not unnecessarily take up space. Details of this design have been further tweaked throughout the development (e.g. the widths and heights of the panels), but the general structure of the latest development version follows the figure below.

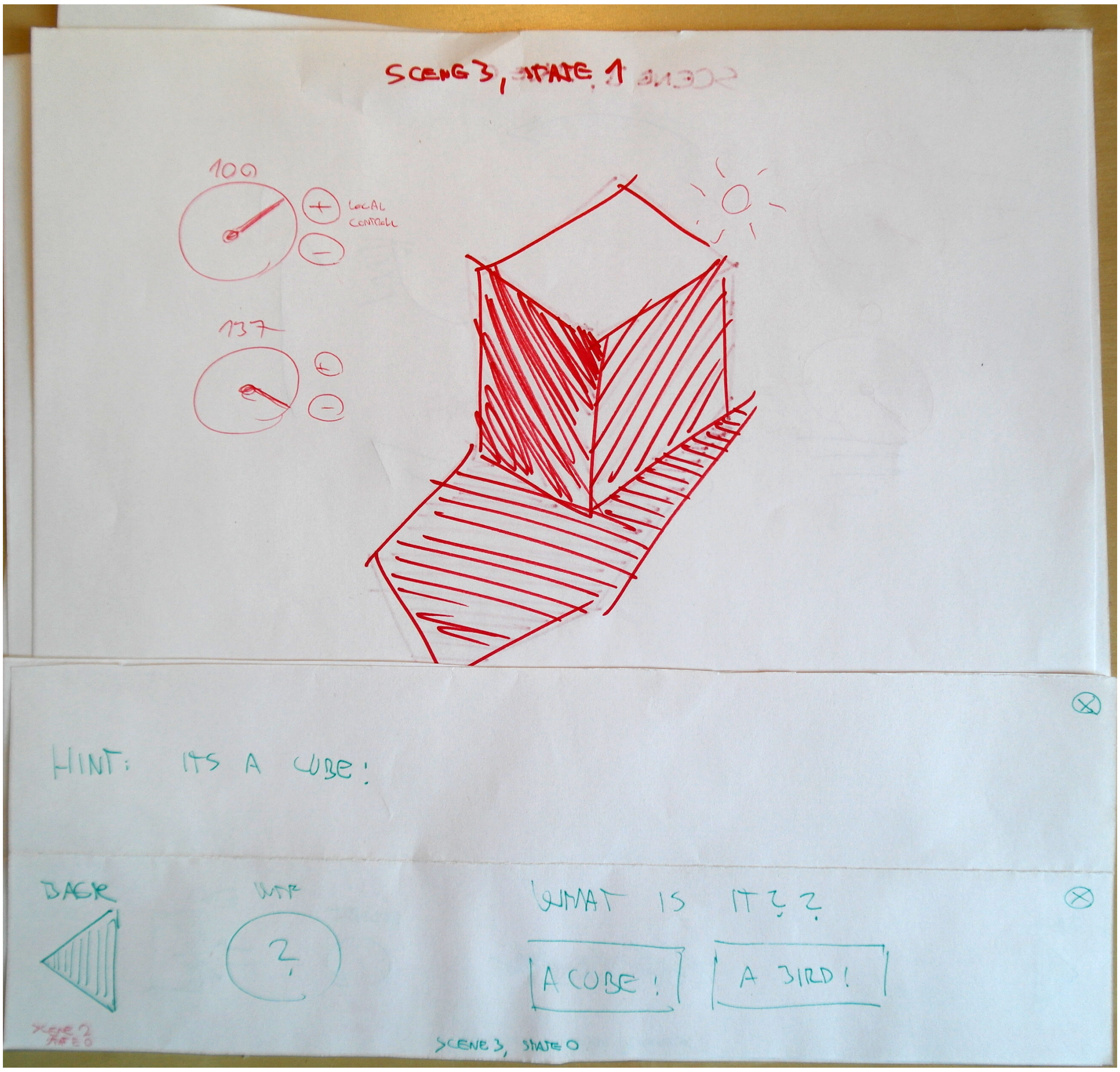


Figure 3.5: Early paper prototype of GUI design.

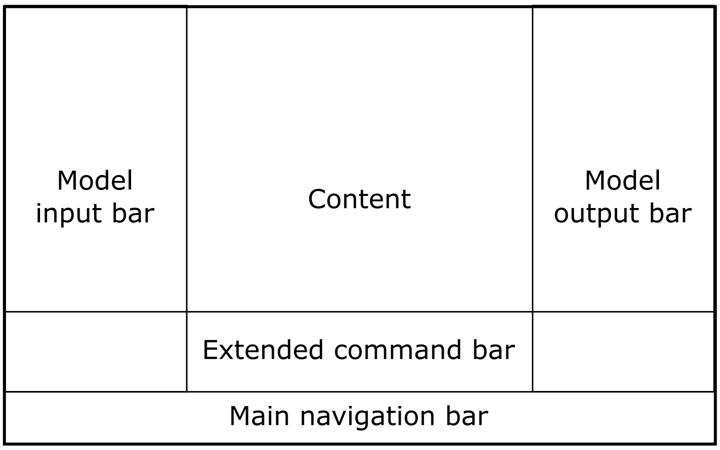


Figure 3.6: Structure of the GUI. The extended command bar is meant to be semi-transparent.

This structure is based on physical prototype and early development GUI design testing. The position of command bars at the bottom of the screen proved to be practical, as people (at least in the target demographic) are used to reading from top to bottom. Therefore, they will reach the navigation bar, and potential questions, after seeing the main content. On the other hand, the inputs and outputs of the model will be displayed side by side with the content, thus keeping the context.

3.3.2 Design features

3.3.2.1 Scalability

Scalability is an important condition, which the GUI has to meet, as the target platforms can vary from big projection screens, to notebooks, and even smaller mobile devices. Even though the main mobile device platform that is aimed for, are tablets (with relatively large screen sizes) the possibility of smart-phone usage is not generally out of the question.

Fortunately, Unity has recently released a UI system, which allows visual positioning and scaling of components. These components can be mutually bound, constrained or use a variety of layout settings. This has allowed a design of the scalable GUI without any greater problem. Responsive design has been researched as well, but it was decided that for the targeted devices can use the same layout, only varying in the component relative scales.

3.3.2.2 Mobile device optimisation

Despite not planning a responsive design, the GUI still has to be adapted to mobile devices not visually, but functionally. Mobile devices usually have fewer buttons (if any) and generally different input methods. These have to be taken into account.

Moving to the next or previous content should be also possible by swiping the touchscreen with a finger in the appropriate direction. This functionality has to be implemented in such a way, that prevents interference with the content itself, and vice versa, manipulating the content must not trigger events leading to showing different content.

In the matter of showing the previous content, it is also appropriate to mention another example of mobile-specific behaviour, which is the usage of standard 'back' button on Android[5] devices. This button, when pressed, should also be bound to the 'show previous content' action, as this is the expected functionality.

The content has to respond to the rotation of the screen, and the user should be able to force the screen orientation as well (when the application is deployed on a device with this functionality). This can be very useful when designing a model visualization with fixed width, while the screen also has to display the input and output panels. Fitting such components into a mobile device in a 'portrait' mode might prove challenging, and in case of a very narrow screen, perhaps even impossible. Forcing a 'landscape' mode on the device can solve any issues that would occur.

Native UI style In this context, it is also convenient to discuss the use of standardized system designs. Each operating system has its own standardized visual style and design patters, that create a unified experience throughout the system. These styles are deemed 'native' in the context of the particular mobile system. Many developers adopt native styles when developing applications for a particular system, or even create multiple visual styles, each for different OS, when developing a multi-platform applications. Articles comparing the mainstream mobile operating systems can be found on-line [9, 13].

Evaluation of the importance of adapting the UI to the native design, has been found in a previously mentioned (3.1.2.1) editorial: "When I made my app in the Android SDK a few years ago I went strictly native for UI design. I wanted my app to look consistent with others. The thing I realized much later was that consistency doesn't matter if nobody else is being consistent. If you look at any of the top apps on the Google Play you'll find that nearly all of them use their own custom widgets/UI rather than 'standard' Android stuff. If anything, I noticed that the 'pros' and big companies designed their own UIs and the hobbyist devs used the standard stuff. So, if anything, I feel like using the standard UI can have a negative connotation of looking amateurish. Anyway, I think the ultimate key is making sure your UI is clean, intuitive, and good-looking... and you don't need to go native to do that."

To conclude, native style is not necessary, as long as the custom UI style is well designed. Creating native design also basically means adopting the API of each system, and developing separate UI for each of them. Both of these tasks would be extremely time consuming, and therefore the proposition is to create one unified, well composed design for all target platforms.

3.3.2.3 Home screen

Including a 'home screen' is a standard practice of giving the user optional actions before showing the main application content. Instead of showing the content, users can find various settings here. In case of this thesis, there are currently no plans of having any user-changeable settings (to promote simplicity), and inclusion of a home screen might therefore

[5] 'the operating system that powers more than one billion smartphones and tablets' - www.android.com/history/

not be necessary or may even be contra-productive, since it is not a common practice on mobile devices.

3.3.2.4 Modularity

Modularity is a feature of the framework. It is vital for its proper usage that programmers can extremely easily customise parts of the UI. This can be done utilising the Unity ’prefab’ feature. Prefab - an abbreviation for prefabricate - is an aggregate of objects (UI elements and attached scripts), that can be multiply instantiated, reused, or attached to another part of the UI using a visual editor. By creating appropriate prefabs, with correctly structured code, a very simple mechanism for visual GUI editing can be implemented.

Prefabs can be used to provide different templates for the content (html/picture/3D), or optional extended command bar content. This bar will hold various types of actions, depending on how the programmer will want the user to interact with the currently displayed content. Proposed types of questions are standard yes/no question, a quiz question (multiple options where one is correct) or a text-input question. The framework will provide prefabs with finished structure, where the programmer only has to specify the question text and answer of the target buttons, and, in case of the textual answer, the accepted text. For example, pressing a button with the correct answer can lead the user to an interactive animation, pressing the wrong answer can show extended explanation, so the student can study why his answer was wrong.

3.4 Identity

To enhance the identity of the developed framework, a concise name was selected: RadApp - as short for Rapid Application Development Application. Since an ’Application for applications’ is being created (or more precisely, a framework for their rapid development). On top of this name, a simple, modern logo (fig. 3.7) was designed. Even though the product of this thesis is not an actual application, but a workflow and tools within existing applications, the final product of the framework usage will be applications, and provided that the developers will not want to create their custom graphics, a default framework logo can be used in suitable places, such as application icons, loading screens or GUI background pictures.

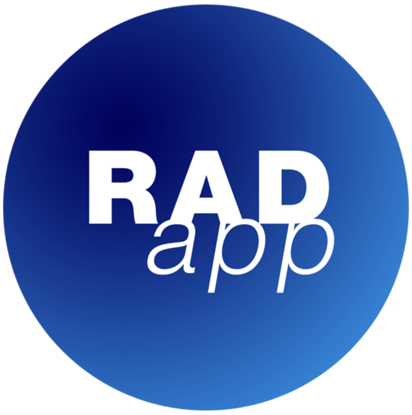


Figure 3.7: RadApp logo.

The logo was designed in such a way, that it can be used as a brand logo, but also as an icon. Since it is round, it fits well with the modern icon styles, and its general simplicity follows the latest trends in logo design. This design can also be easily scaled to small proportions that can be deployed on mobile screens. "That logos have to be scalable has always been understood. But our perception of ’small’ has changed, in some cases ’tiny’ is being rather generous. Dimension and detail are necessarily removed so that these logos read properly on mobile screens. Designs have become more and more flat. Surfaces are plain and defined by mono-weight lines."[6]

The following images depict the use of the designed logo as a Windows desktop icon (figure 3.8, and as an icon in the 'Apps menu' and on the Android home screen (figure 3.9). Both environments show the icon side by side with the operating system standard icons, to demonstrate the compatibility with the overall visual style. Stock backgrounds were used for these screen-shots.

As the logo visually fits on both operating systems, in different resolutions, the design can be judged as successful . Unity has a native menu where it is possible to set icons for all available platforms, that support icons, and this logo can be this way, so the icons are always automatically deployed with each build for every platform.

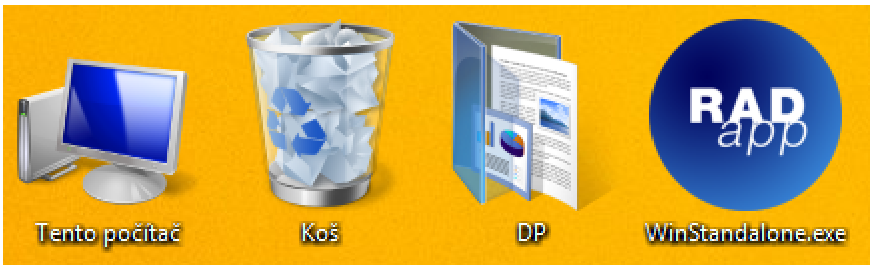
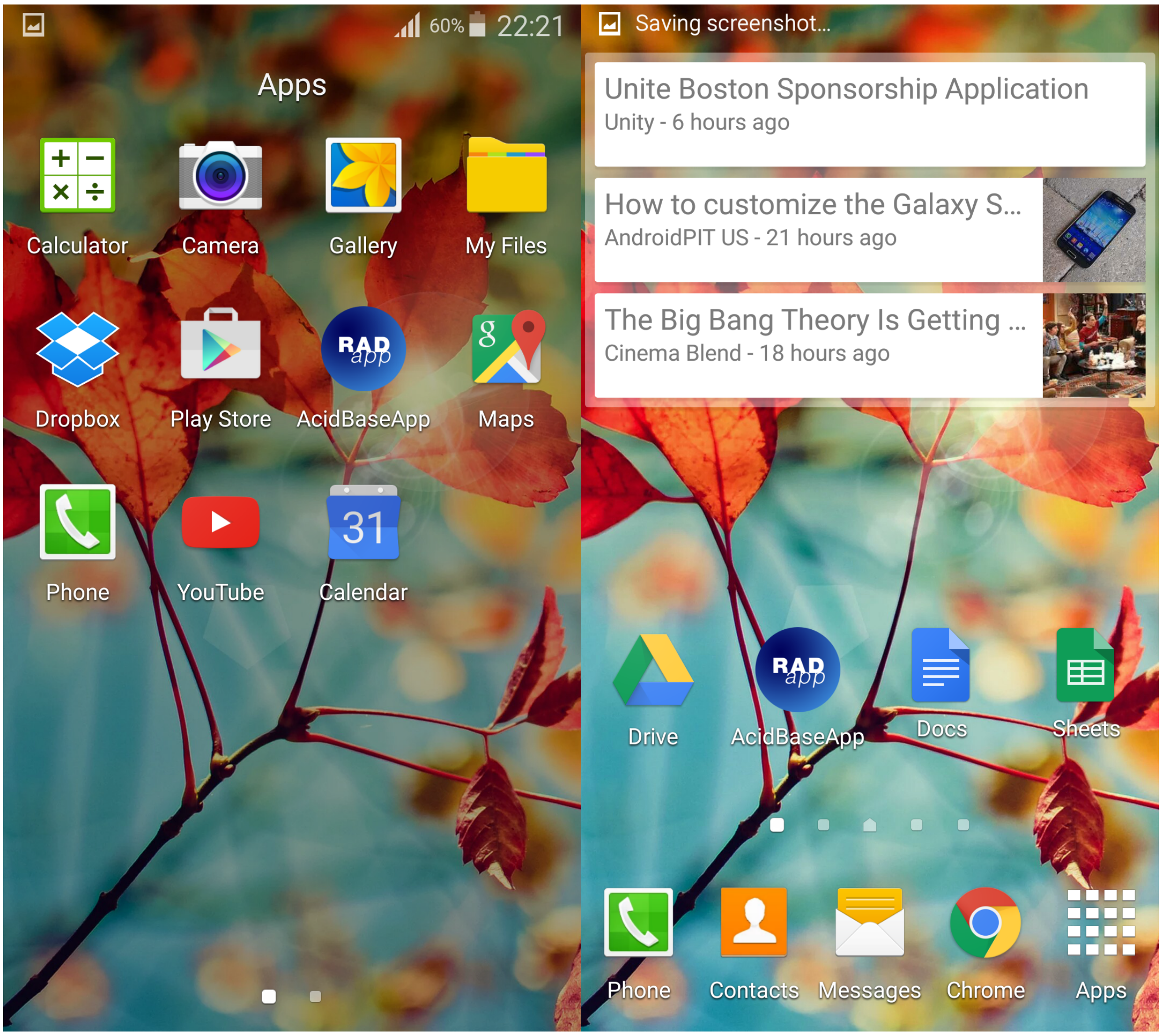


Figure 3.8: RadApp logo as a desktop icon on Windows 8.1 operating system, next to a set of standard icons.



Unite Boston Sponsorship Application
Unity - 6 hours ago

How to customize the Galaxy S...
AndroidPIT US - 21 hours ago

The Big Bang Theory Is Getting ...
Cinema Blend - 18 hours ago

Figure 3.9: RadApp logo as icon for app deployed on a smart phone with Android 5.0.1 operating system. Left image: RadApp icon amongst standard Android apps in the 'Apps menu'. Right image: RadApp icon on the 'Home Screen', depicted again with other icons in a common setup.

Chapter 4

Realisation

In this chapter, the implementation process and its results, along with the solutions of encountered problems, are presented. The implementation itself is divided into two logical units. Firstly, the work done in Unity environment, and secondly, the work done on Bodylight framework. The finalisation of the latter also included some minor implementation in Unity, when the interconnecting interface was developed.

4.1 Development environment

Before getting into the implementation process, it is reasonable to present how the development environment was set up, what supporting tools have been used, and how the implementation process was managed.

The main tool is, unsurprisingly, the Unity software, which also comes with embedded Mono Develop code editor. Originally, the Unity version 4.6 was used, but during the development Unity 5.0 was released, and the project was upgraded to this version, in hopes of solving some deployment issues.

For the Bodylight development, Visual Studio 2013 is required, along with several plugins and additional libraries. Fortunately, Visual Studio manages their deployment automatically. The Bodylight framework source is then accessible on a private git server, which requires access from the members of laboratory of bio-cybernetics, 1st medical faculty of Charles University in Prague.

Already mentioned is another extremely useful tool, which is the git project management system named Git[1]. This system allows a simple version control and code changes management. This has proven useful when producing a demonstration application, while still continuing the development of the framework.

An issue tracking document has been set up on the Google Drive[2] online file storage, in a form of shared spreadsheet. In this document, the development tasks were noted with current status. This, along with git, has allowed the thesis supervisor to regularly check the progress and maintain an overview of the work.

[1] 'a free and open source distributed version control system' - git-scm.com

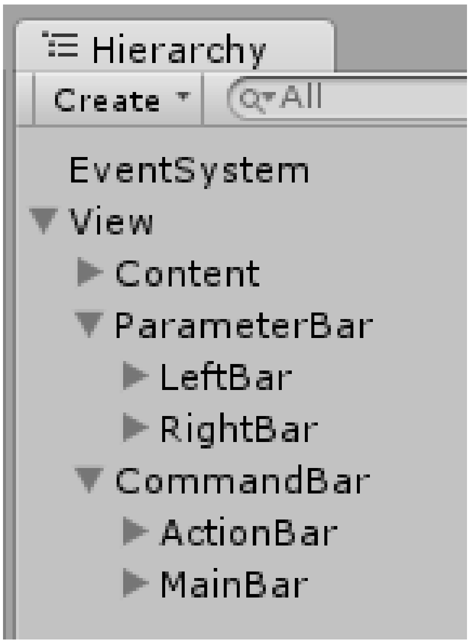
[2] Online file storage and office system by Google inc, available at www.google.com/drive/

4.2 Implementation in Unity

4.2.1 GUI Layout

The first task in Unity was to create the proposed GUI structure and its basic functionality. Setting up the components in a correct layout was easy thanks to the Unity UI component system. The components proposed in section 3.3.1 were composed in correct order, and appropriate controllers were created. The hierarchical structure of the visual elements is shown in figure 4.1. The finished GUI structure is shown in picture 4.2.

The proposed home-screen has originally been created, and automatically added to the final applications, but has proved unnecessary, and even annoying (especially on mobile devices). There are no user settings, only start and exit button. On most devices, it completely lacks purpose - closing the application is done by 'home' button (the middle button on both android and iOS devices) or desktop window buttons (usually the top right corner of the window. The extra screen only inserts more time between starting the application and showing the content, without any added functionality. Therefore, in the current version the home screen is disabled.



```
Hierarchy
Create ▾   Q▾All
  EventSystem
▼ View
  ▶ Content
  ▼ ParameterBar
    ▶ LeftBar
    ▶ RightBar
  ▼ CommandBar
    ▶ ActionBar
    ▶ MainBar
```

Figure 4.1: Hierarchical structure of the main GUI elements. EventSystem is a Unity component required for correct control of user input, and does not have a visual representation.

4.2.2 Drag and Drop system

A key feature that Unity provides is a drag-and-drop system, that allows the user to connect components without explicitly writing any code. Custom scripts can be attached too, as long as they extend the MonoBehaviour[3] base class.

By extending the aforementioned class, Unity automatically detects class variables with public access, or with 'SerializeField' attribute. When the script is attached to another component, these variables are displayed in the inspector window, along with editable fields where user can specify their initialisation value. Example of this behaviour is shown in picture 4.3. The relevant code of the attached script follows in the figure 4.4.

[3] 'the base class every script derives from' - docs.unity3d.com

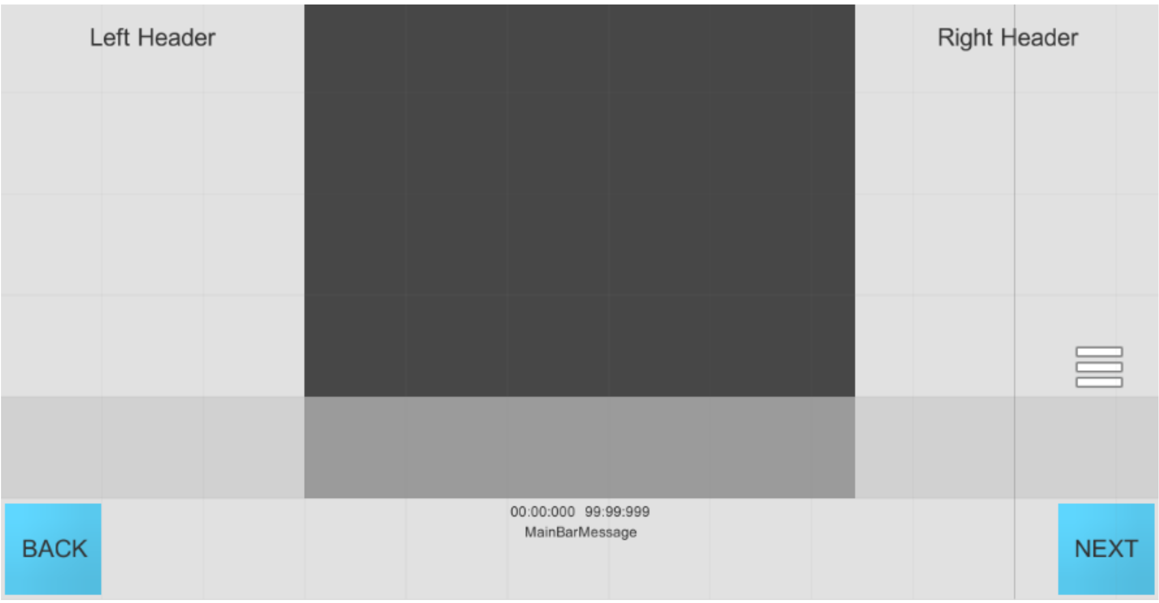
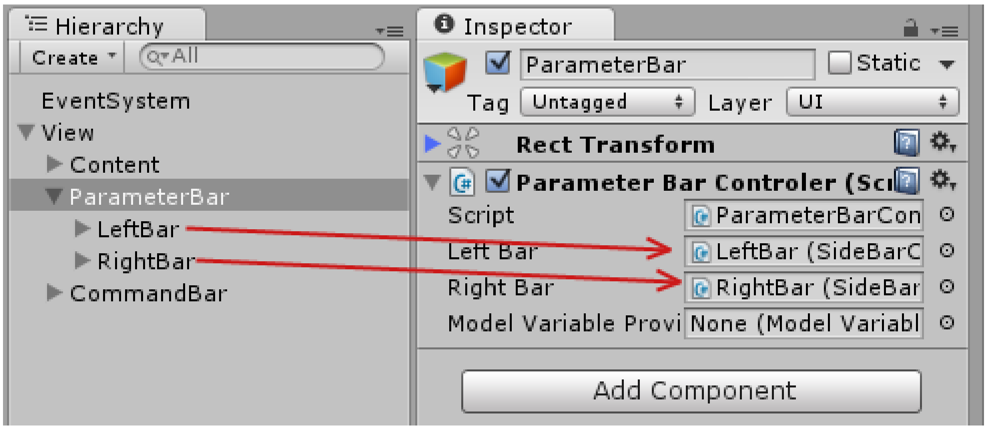


Figure 4.2: Finished GUI structure.



ParameterBar ☐ Static

Tag Untagged Layer UI

Parameter Bar Controler (Sc

Script ParameterBarCon

Left Bar LeftBar (SideBarC

Right Bar RightBar (SideBar

Model Variable Provi None (Model Variabl

Figure 4.3: An example of drag and drop assignment. The red arrows shows which components had been dragged and where they were dropped. The 'LeftBar' and 'RightBar' rows in the 'Hierarchy' window, represent objects that are present in the scene, and that have a 'SideBarControler' class attached. The inspector automatically recognises the class required by the variable that is being assigned, and does not assign the dropped object directly, but its component of appropriate class.

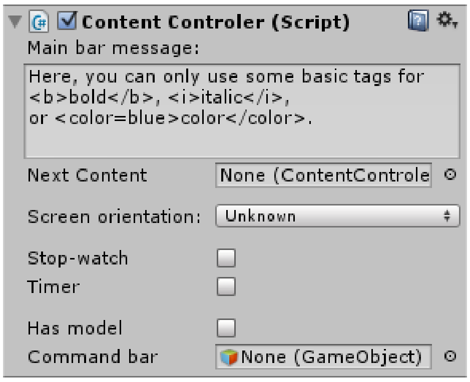
```
public sealed class ParameterBarControler : MonoBehaviour {

        public SideBarControler leftBar;
        public SideBarControler rightBar;

        public ModelVariableProvider modelVariableProvider;

        // ...
        // the class body
        // ...
}
```

Figure 4.4: The code of ParameterBarControler class, whose inspector is shown in picture 4.3. Note the access modifiers and the type of the public variables.

This method of assignment does not only work for objects present in the scene, but also for prefabs, which proved useful when creating a simple system of modular action bar. This is further described in section 4.2.5.

4.2.3 Adding content

Creating a new content in the application is very simple. The user creates a new object in the Content section of hierarchy, shown in figure 4.1. This object is then assigned the ContentController script, which creates the necessary interface. The user can then proceed to create his own content in the object. Another option is to create new instance of one of the prepared content prefabs, which are discussed further in section 4.2.6. For the full procedure of creating and linking content, please refer to the attachment B, 'RadApp user manual'.



☑ Content Controler (Script)

Main bar message:

Here, you can only use some basic tags for <b>bold</b>, <i>italic</i>,
or <color=blue>color</color>.

Next Content: None (ContentControle

Screen orientation: Unknown

Stop-watch ☐

Timer ☐

Has model ☐

Command bar: None (GameObject)

Figure 4.5: The Content Controller custom inspector.

4.2.4 Custom editors

Unity also allows a custom inspector window for every MonoBehaviour extended class, which is extremely useful when presenting advanced types, or when aggregating multiple classes in one window. This feature allows creation of custom editor components, instead of the automatically generated fields for public variables, as shown in picture 4.3.

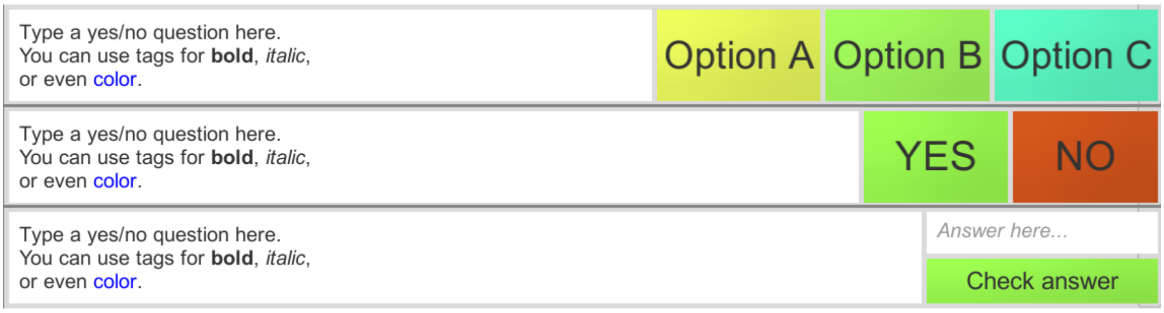
Content controller custom inspector will be the main interaction component for most developers. It aggregates all relevant settings into one interactive window. As shown in figure 4.5, the basic component provides several settings, that can automatically show and initialise components of the GUI when the specific content is displayed. The most important part here is the 'Next content' field, where user can specify which content will be displayed after pressing the 'next' button in the application GUI. After assigning this property, another field - 'target state' - will display, where user can optionally also set the target contents state. For an extended explanation of each propriety, see attachment B, 'RadApp user manual' and for the description of the ContentController class, see attachment C.

4.2.5 Action bar content

As proposed in the Modularity paragraph of section 3.3.2, prefabs were used to create several types of action bar content. These prefabs can then be assigned to the 'Command bar' field of the content controller via the inspector window (see figure 4.5).

There are 3 provided actions as of yet, with the interface being done in such a way that allows its easy extension to new types, or similar types with more buttons. Below is a list of currently provided types, and figure 4.6 presents their look when deployed in the GUI.

- A/B/C quiz question
- Yes/No quiz question
- Text-input answer



Type a yes/no question here.
You can use tags for **bold**, *italic*,
or even color.

Type a yes/no question here.
You can use tags for **bold**, *italic*,
or even color.

Type a yes/no question here.
You can use tags for **bold**, *italic*,
or even color.

Figure 4.6: Prepared action bar content. From the top: an A/B/C question, a Yes/No question and a Text-input question. The labels of buttons of the first type can be changed by the developer in the Unity inspector.

Assigning these prefabs to the appropriate field in the Content Controller inspector extends the inspector to show a new set of options, where user can type the desired message, specify target content of each button, and optionally the target content state. In case of the text-input question, user can also specify multiple accepted answers. In case of the A/B/C question, user can also change the labels of the buttons. This system is also created in such a way, that creating new types of action content is a matter of minutes. See C for further information about the implementation.

4.2.6 Content prefabs

Another use of prefabs is to provide users with an easy way of setting up new content of the same type. In case of the default environment, only one such prefab exists, and that is for the html content. When creating an application with advanced content, for example a 3D scene, the user can create a prefab of this content, and then very simply reuse it while keeping the very same structure. Since it is not currently possible to link one content to itself, even if the state parameter changes, this method of storing the structure of a custom content has proved very useful when creating the example application (see attachment A).

Even if no further prefabs of content are present currently, some of the utilities are provided in the form of a prefab. Such as the static graph, which has a complicated component structure, and its deployment would not be as easy as assigning the appropriate controller to a scene object. There is also an example of links usage in an HTML content (see section 4.2.7).

4.2.7 HTML content

One of the most time-consuming parts of development was deploying a tool that would provide support for showing HTML files and HTML formatted text. For reasons described in 3.1.3, HTML was the proposed solution for displaying longer texts. Unfortunately, Unity did not release any update on native HTML formatting support in the duration of this work, and it is unclear if such feature will ever be included. For this reason, an external tool had to be incorporated.

4.2.7.1 Awesomium

Awesomium is a conditionally free plugin, with specific support for Unity. After looking for available Unity plugins, and an unpleasant fact that official plugins require the professional version of Unity, the set of options became very limited. Awesomium came out as one of the options that, in one point, worked nicely in the Unity environment.

The downfall of this plugin was caused by several problems during deployment. Although the build process usually finished successfully, the application failed to display any HTML content, and crashed shortly after running. Even the desktop builds were failing, and it was later discovered that there is no official support for mobile devices. Discovering this fact took away any motivation to further try to overcome multiple difficulties Awesomium has presented, and the plugin usage was discontinued. This has also removed the only available option of displaying video content, since the native video support is only available in Unity professional, Awesomium could have provided videos through HTML 5 (This matter is further discussed in section 4.2.11).

4.2.7.2 HTML parser

For the majority of time after discontinuing Awesomium, the framework was without any HTML support, and as the development of other features begin to finalise, it almost looked that the final release for this thesis wont be able to utilise HTML formatting. Fortunately, when all the other major problems were dealt with, the issue of HTML has been reopened. Instead of looking for an HTML plugin, which usually led to an extensive engine for displaying live web pages, the search conditions were changed to look for HTML parsers. Several discussions were revealed, in one of which an open source HTML parser called ’HTML engine mini’[4] has been linked.

This independently implemented HTML parser is fully written in C#, and therefore can be directly included in Unity. Since its development has ceased in June 2012, several modifications had to be made to be compatible with the current UI system of Unity. It is a very basic parser, with only a limited subset of supported HTML tags, but is sufficient for the purpose of this work. As it is open source, it can be further extended to support more required tags, but currently the support of basic text formatting and displaying images is all that is needed.

Currently supported tags with basic syntax examples (parameters in brackets are optional):

Link -	<a href=’link’>link text</a>
Image -	<img src=’imagesource’ [width=”] [height=”] />
Paragraph -	<p [align=”][valign=”]> </p>
Line break -	<br/>
Font format -	<font [face=”] [size=”] [color=”]>
Code format -	<code></code> same as <font face=’code’>
Bold -	<b></b>
Italic -	<i></i>
Underline -	<u></u>
Strike -	<s></s>
Text effect -	<effect></effect>
Division -	<div></div> - this element does not always render correctly.

4.2.7.3 Linking

The parser also provides methods for creating functional links. Moreover, it leaves handling of the links on the developer, which means that the parser returns the address of the link, and a custom code can then handle it. For this purpose, a custom event has been created, that is invoked when the user has clicked on a place on the screen where a link tag is displayed. This event can be observed from any user script, and then handled the same way as any other button click event. The target parameter of the HTML link is passed through

[4] Open-source ’Device independent HTML parser and render for any .net/mono graphics toolkit’ by Ruslan A. Abdrashitov - code.google.com/p/html-engine-mini/

the event, and can serve to decide which content to show, or how to manipulate currently the displayed content. This functionality is again described in detail in the attachment C, and its practical usage in attachment B. Furthermore, a prefab with an example use of the HTML linking is also provided.

4.2.8 Input and output bars

The left and right parameter bars on the screen, as visible in picture 4.2, are utilised only when a model is attached to the content (the model connection process is described in section 4.3.2). The bars have separate functionality, as the left bar serves for inputs, and the right bar for outputs, as proposed in the design section 3.3.1.

There are several types of variables that can be displayed:

1. Interval
2. Select
3. Boolean

All of these can be seen in the picture 4.9, in the left command bar (as they are input types). The interval variable provides floating-point number value. The developer can specify minimum and maximum value, disable the slider on the run, or change its output to whole numbers. In the attached picture, these are the two top slots, labelled c(H) and c(OH). The select value examples are the 4 following slots, and serve to select one option from a mandatory number of options. The selected option is highlighted, and returns an integer number, representing the index of the selected option. The last type is boolean, which is a simple toggle button, that can either be true or false.

All of the input types can be also manipulated from the model-side. This means, that even though they are designed to serve mainly for input, they can be bound to reflect the current values in the model. E.g. in the example application, the user drags the slider to the desired value, and when he drops it, the model disables its use and drags both the sliders to reflect newly calculated equilibrium. After settling on a new value, the sliders are enabled again.

The output variable types are:

1. Output
2. Text output
3. Text output with graph

The first item, an 'Output' type variable only serves to hold a float value, but has no visual component. This is useful if the developer needs to store some values for his custom animation, that do not need to be displayed in the left bar, as they are not an

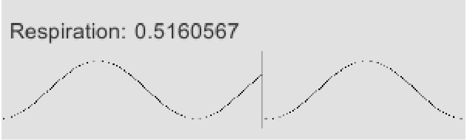

Respiration: 0.5160567

Figure 4.7: The visual component of the graph output variable.

interesting in the current context (for example values that specify some object's position in the animation). The text output shows specified text. The 'text with graph' output shows a graph of a numerical value changing in time. The user has to specify the scale of the value, so it fits into the graph vertically. Horizontal scale is then set by binding each pixel to one rendering frame (this is an unfinished behaviour, as it should be bound to actual real time). Example of this slot graphics is shown in figure 4.7.

The class hierarchy of the variables is shown below in the figure 4.8. The ModelVariable is an abstract class that provides only the basic interface for manipulation in low level logic. Each derived class provides further functionality, and has its own slot type (except for the ModelOutputVariable class). The visual components are again created as prefabs, that are instantiated when a new instance of the variable class is created. The base ModelVariable class does not derive from the usual MonoDevelop base class, since it is not a component that can be added to any visual object. The instances of variables have to be created in code by the developer, when connecting the Modelica model, as described in 4.3.2.

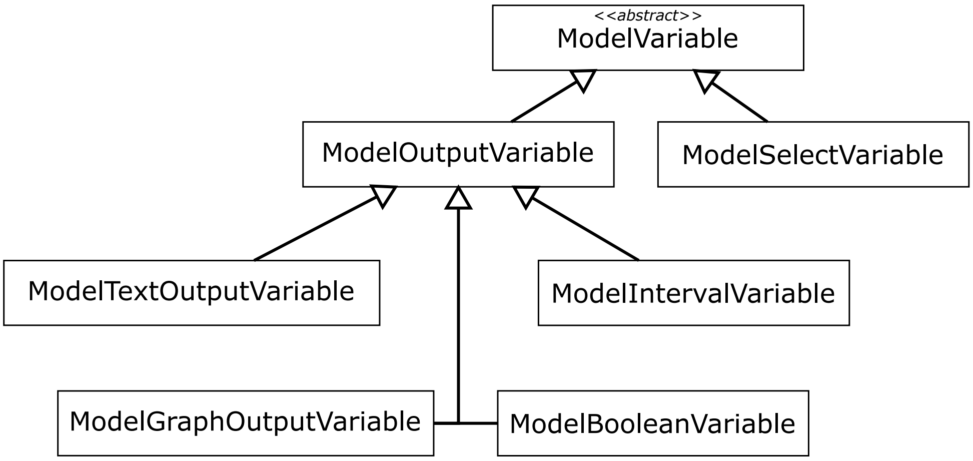


Figure 4.8: Class hierarchy of the variable types.

4.2.9 Platfrom-specific behaviour

As proposed in the design section 3.3.2, some specific behaviour for mobile devices has been implemented. As proposed, navigating between contents is possible by swiping the screen from right to left to go 'back', or in the opposite direction to show 'next' content. The standard android 'back' button has the same signature as desktop 'escape' keyboard button. This functionality did not seem out of the ordinary, and so both mentioned buttons trigger the 'show previous content' action. Apart from utilising touch screens for navigation between contents, and using the standard android buttons, some other visual tweaks were made to make using mobile devices more pleasant.

One of such adjustments is scrolling through content exceeding the screen, which, on touch screens, is intuitively done by sliding ones finger on the screen in the appropriate direction. On desktops, where touch screens are not common yet, sliding is either done by mouse wheel, or by dragging a slider located traditionally on the right side of the content.

This slider is automatically present in the HTML content, and is also automatically hidden on mobile devices. This also requires repositioning the 'hide command bar' button to the edge of the screen.

The application also automatically detects mobile devices (currently iPhone and Android) and adjusts the HTML font size accordingly to the device dpi. This is a feature that allows the HTML content to be displayed correctly even on smaller smart phone screens, although the images within the text still remain relatively small, because the technical solution only affects text. Even most of the controls were adapted in such a way, that with some small effort they are usable on smart phones, but for example the interactive variable sliders (see picture 4.9), are already oversized on desktop screens, but still very difficult to use on phones. Basic non-model applications can target mobile phones without any larger issues.

4.2.10 Options for vector graphics

Unity does not natively support vector graphics. Free solutions are strongly preferred, and by so, they are limited to the free version of unity, and also forced to disregard the use of paid plugins or assets for unity, which also usually require the pro- (paid) version of Unity (see 4.2.11). There are several general options of handling the problem:

1. Saving the graphics in a vector format, and find a way to display this format in Unity.
2. Exporting the graphics as a raster, and using this picture as a simple texture in Unity.
3. Saving the graphics as a vector, using other software to convert it into a 3D object, which can then be imported to Unity.

In further paragraphs, these options are discussed with the advantages and disadvantages of each solution.

4.2.10.1 Displaying vectors in Unity

This is a straightforward solution to the problem, that completely removes any necessity to manipulate with the graphics outside of the program used to design the graphics. Since the Unity renderer always uses the actual size of target texture, the image keeps all benefits of vector-based graphics. As stated in the problem definition, the main issue with this approach is, that Unity does not directly support a vector-based graphics. There exist plugins that solve this issue, which are either paid, or with less functionality.

A paid solution is purchasing a plugin enabling this functionality. For example SVGAssets[5] plugin reads directly from an SVG file, and renders it on-the-run to provided texture surface.

Advantages: The plugin is finished and optimised, professional product, which guarantees its proper functionality and performance.

[5] Plugin available from Unity asset store at assetstore.unity3d.com

Disadvantages: Is not free. The plugin itself costs $125.00, and requires Unity Pro. Furthermore, it has limited platform support: Web Player, Windows Phone 8 and Windows Store are not supported.

To conclude, this is a clean and robust solution, preserving all aspects of vector-graphics, but also creating either financial demands, or deployment problems.

A free solution is an open-source alternative, called UnitySVG. As the author himself states on the project page, "A very limited subset of SVG is supported. (...) Essentially, only things involved in rendering static images are supported, with no regard to clarity of the XML, 'reuse', or whatnot"[6]. There are also some performance issues, and the plugin itself is not currently being updated (last update to the code is from 4. October 2012).

Advantages: Free and open-source. This gives an option to manipulate the plugin core directly and possibly change its functionality to fit any needs.

Disadvantages:

- Unfinished side-project. The author created this plugin to fit his needs and is not developing it anymore. IT can be unstable, unpredictable, and the last tested version of Unity is 3.5.6.
- There is no mention of platform support at the project page, which would require further research.
- The deployment of this solution could cause compatibility or performance issues.

This is more of a work-around solution, since use of raster pictures defeats the purpose of creating a vector in the first place, even with the use of the Unity sprite capabilities. It is, however, the simplest and fastest possible solution.

4.2.10.2 Rasterising vector graphics

Exporting the graphics to a raster format is the easiest way of handling the problem, but can also be considered as more of a work-around, since it defeats the purpose of creating a vector graphics. Unity natively supports a variety of raster images. Photoshop .psd and .tiff are imported with flattened layer. Multiple formats (JPEG, PNG, GIF, BMP, and many more) are supported.[7] The format support overlaps with the capabilities of most used vector graphic programs, which provides a simple solution, at a cost of loosing the advantages of keeping a vector format. Moreover, Unity has a very useful sprite editor, which allows creation of scalable graphic, without loss of sharpness on its edges. Its function is limited, but provides some basic features we look for in the vector format. There is also a plugin[8] that simplifies the workflow, but it is not free, which greatly exceeds any possible advantages.

Advantages:

[6] Cited from the project page - github.com/MrJoy/UnitySVG

[7] Adapted from unity3d.com

[8] Illustrator->PNGs, available from Unity asset store at www.assetstore.unity3d.com/en/content/25491

- Simple and ready. There is no need to learn or deploy anything, we only use what we know, and what works.
- Can be edited as a sprite and make use of the Unity UI components, which means easy drag-and-drop scene composition.
- It is completely free.

Disadvantages: Does not keep a vector nature of the graphics. This solution completely defeats the purpose of creating graphics in a vector format, and subsequently creates problems with the graphic scaling or rotation.

This solution allows manipulation of the graphics in Unity without significant loss of quality, and is very easy to implement, but also removes the simple drag-and-drop approach in the Unity 2D scene, that is only possible with raster images.

4.2.10.3 Creating a 3D mesh

This is a compromise between solution 1 and 2, since it keeps some features of the vector format, but it is also very easy to deploy. By using a free software[9], this solution stays cost-free, but is slightly more time-consuming, since the graphics has to go through a manual conversion in yet another program.

Advantages:

- Still relatively easy to do. It creates one extra step in the export-import workflow between graphical software and unity.
- It does not keep all the vector specifications, but because it is a 3d mesh, unity can scale and rotate it without further loss of quality. There is some discretization of vector lines necessary, but the level of detail can be adjusted accordingly.

Disadvantages

- Creates a 3D object. Its use is not as straightforward as with a simple raster image, and it requires a bit more computing power, although this should not be an issue on a current-generation devices.
- Only useful for outlines and UI graphics without complex inner structure.

4.2.10.4 Conclusions

As the first option (4.2.10.1) would create extra demands on time or finances, and both are preferred to be kept at minimum, it was not be further researched.

The second option (4.2.10.2) was used for simple graphics. In sufficient resolution and in combination with the Unity sprite editor, designs of high quality can be achieved. The possibility to use these sprites within the Unity UI system is also a strong argument for this solution.

[9] Such as Blender - www.blender.org

Finally, the third option (4.2.10.3) can be used to create shapes that can be manipulated both structurally and visually. Since unity has an animation interface, predefined movements can be created for such shapes. Unfortunately 3D objects can not be interconnected with the Unity UI components, and so it can be a little more problematic to properly incorporate them with flat UI elements. This approach was researched and is entirely viable, but for the example applications it was not suitable, and so it remains unused.

4.2.11 Advantages of professional version

During the development of the Radapp framework, several problems were encountered, that would be easily solved by using the professional edition of Unity. One of the main goals is to keep the development as independent and cost-less as possible, and thus, it has to be refrained from choosing the financially more demanding solution, and a workaround solution using only free tools is preferred. That being stated, some of the encountered problems were not solved in a reasonable amount of time, or the workaround would be too complicated for a task too trivial (see the list of unfinished tasks in the section 5.1.9). For these reasons, the option of purchasing a professional licence for a limited amount of time, or using a trial version of Unity to solve some of these issues, is still an open option, but was not utilised it this thesis. Following is a list of advantages that the professional version of Unity would provide.[10]

4.2.11.1 Video content

This is an example of a problem, for which a solution was not found. In the professional version of Unity, there is a very simple way of playing videos as a surface of any object in 2D or even in 3D. The video is simply passed to a MovieTexture component, which supports several video formats. On Android and iOS the videos have to played in fullscreen, using function Handheld.PlayFullScreenMovie[11], which is also a pro-only feature. This means that video content would have to be created separately for desktop and handheld devices, but Unity does have direct code support for such cases[12].

There was one promising way of dealing with this issue, and that was through displaying the video via a HTML5[13] tag in a displayed HTML page. This possibility is discussed in the next section. The Unity Professional edition would provide native video support.

4.2.11.2 Advanced HTML content

There is a very complex plugin for unity, that adds the ability to use web page directly as a texture in unity, called Awesomium[14]. Using the chromium engine makes it possible to display any live web page from the internet. The plugin has specific releases for Unity, and has even implementation examples and tutorials available. The plugin is free for developers with less than $100k annual revenue.

[10] Full list of features not included in the free version can be found on unity3d.com/get-unity

[11] See docs.unity3d.com for the code reference and manual.

[12] Platform dependent compilation, a feature of C# programming language.

[13] A next generation specification of HTML that is currently in development.

[14] www.awesomium.com

There are several issues with using this plugin in the free version of unity. Mainly, the finished application does not seem to find the proper Awesomium libraries, and fails to deploy. It is questionable, whether this is caused by the free version of Unity, but when running the example scene, the editor throws several errors regarding a missing professional licence, which hints in this direction.

Nevertheless, the biggest problem of this plugin is, that it does not support mobile devices, but only desktop Windows and Mac, which fails the target to stay as multiplatform as possible.

There are several other solutions to display an HTML page as a texture, but they are either unfinished discontinued projects, or are also limited to certain platforms. It is uncertain, whether the Professional edition of Unity would help support online content. And even if it was the case, it might still be limited to few selected platforms.

4.2.11.3 Custom splash screen

In the free version, on any platform, a default Unity logo is displayed for few seconds, before the program itself is loaded. In the professional version the picture can be changed to whatever desired - for example the RadApp framework logo (see 3.7).

4.2.11.4 Plugins and core code access

The core code of Unity is inaccessible in the free version, and since all official plugins are based on manipulating the core directly, all of them are unusable by default in the free version.

4.3 Bodylight adjustments

4.3.1 Compatibility issues

Bodylight is being developed in the .NET framework, but because its original target platform is Silverlight and WinRT, it was utilising the .NET 4.5 version, and the core was targeted to the Portable subset. This was the main issue when connecting Bodylight to Unity, because the embeded MonoDevelop only supports .NET 3.5 (Generally[15]), and the supplied libraries have to be targeted to full .NET 3.5, instead of the portable subset. Reasons for this are not completely clear, but extensive testing has proved that the biggest issues are caused by the parts of Bodylight written in the F# language.

Most of the incompatibilities were resolved by creating a new build configurations, which included different versions of libraries (especially fsharp.core), and introduced new constants that were then used for conditional compilation directives in the code.

Few lines of code in the core of Bodylight had to be rewritten completely, because of the differences between version 4.5 and 3.5 of .NET (missing methods or classes). Apart from the core libraries, there are application specific classes that are used to connect the model to

[15] See http://docs.unity3d.com/410/Documentation/ScriptReference/MonoCompatibility.HTML for detailed list of compatible libraries.

the view (the ViewModel part in figure 3.2). Some parts of this package had to be completely recreated for use in Unity, especially the ViewServiceImpl class. This class is implementing the ViewService class (for the class description, please refer to attachment C), that takes care of synchronizing the animation framerate with the simulation derivation speed. This has been solved by creating a static UnityTickProvider class, that can be referenced from within Unity, and sends a signal to create a new derivation step. To properly synchronize simulation to the framerate, the static ProvideTick method has to be called from Unity. This is further described in the following subsection.

4.3.2 Connecting model

This is only a rough explanation of principles regarding the realisation of Bodylight-Unity interface. For a complete guide of how to embed a model to the developed application, see attachment B - "RadApp user manual".

Connecting a model requires deeper programming knowledge, since it is the only part of development using RadApp framework, where user has to directly write code. Since Modelica is also a programming language, it is expected that people who create models, and might want to include them in an application, do posses the necessary skills required to carry out the tasks described in the following paragraphs.

For the connection of a model, the ModelVariableProvider interface is provided, which the user has to implement to communicate with the Bodylight ViewModel package. This interface then serves for automatic distribution of user-created variables in the rest of the system, including the GUI.

The connection happens in two layers. Firstly, as mentioned in the previous subsection, through the UnityTickProvider static class. The user should call of ProvideTick method in the initialisation Start method, and the periodically called Update method. These methods are part of the MonoBehaviour base class, that ModelVariableProvider extends. Its implementation then has to be attached to the content (a unity GameObject with assigned ContentController, with HasModel property set to true), in which the model shall be presented. The attachment can be done by compiling the Bodylight parts into a library, that can then be included in the Unity project assets. The code manager automatically adds references to these libraries, and they can then be referenced from code.

After doing so, the user has to manually connect the variables from the Bodylight ViewModel, to the provided ModelVariable implementations. This initialization should take place in the Initialise method, which is automatically called when the appropriate content is displayed.

All of the interactive visual components for the model variables provide an event, that is invoked every time user interacts with the GUI of the specific variable. This event also provides the new variable. In the implementation of the ModelVariableProvider, the user should implement listeners to these events, so that he can trigger new simulation when necessary. Regarding the changes that happen in the model, these should be read in the Update method, where the appropriate model variables values can be updated. The important thing to keep in mind is, that there are two sets of variables. One provides the user inputs, and thus can not be directly connected to the model variables in the Update method, since these constant updates would overwrite the user's input. The second set provides model outputs.

The values from the output variables can be also used to manipulate the input values, but after some sort of trigger, so the model values do not prevent the user input.

These principles are better understood from the examples provided in the code, especially the AcidBase application that has been released to specifically demonstrate the possibility of connecting a relatively simple model.

4.3.3 Custom visualisations

There are several methods of creating a custom visualisation for the connected model, or even simple animations without any model in the background.

Creating a 3D animation is easy by utilising the strong Unity graphical engine, that even includes advanced features for game creation, such as particle effects, water, skyboxes or simulated gravity. The same goes for 2D animations, as Unity can also be used for 2D game development. All these principles are a matter of understanding the Unity environment, and therefore will not be clarified in detail here.

Unity does not natively support vector graphics, which is a slight disadvantage when creating scalable components. In the created demonstrative application, the simple raster graphics in combination with the Unity UI components has been used. Other possible methods were discussed in the section 4.2.10

If the animation should be connected to the model, accessing the variables is simply a matter of referencing the ModelVariableProvider from a custom script attached to the content, that can also reach the animation components. This can also be studied in the attached example application.

4.4 Deployment

Before deploying the application on a selected platform, there are several settings the developer should check first to prevent unnecessary problems.

Pictures that are imported to the resources folder should have appropriate Max Size selected. If building for large screens, select a size that is big enough, so that the picture does not have to be upscaled (given that the original picture does have appropriate large dimensions). On the contrary, if building for smaller screen platforms, select resolution small enough, so that the image does not take up too much memory, which would go to waste because the picture will have to be downscaled to fit the screen anyway.

There used to be some trouble with images in the HTML parser, which are solved in the version released for this thesis purposes. In this version, most of the available settings of imported images should work, but the recommended settings, which are tested are: 'Sprite (2D and UI)' as the texture type, pivot in the centre, and format 'truecolor'. These settings were tested on both desktops an mobile devices and proved to be stable.

4.4.1 Known problems

4.4.1.1 Html parser

The most notable issues are related to the Html parser. Because of how the rendering of final document works, all of the text and images are aggregated into one large texture. If the text is too long, the dimensions of the texture can cause problems on mobile devices. While testing the example application on a Samsung Galaxy S4 smart phone, with Android 5.0.1 operating system, the last 3 pictures in the first content ('Introduction') had to be disabled, in order to render it correctly in the portrait mode. Further research gave a maximum of 95 lines of text in the current settings can be displayed. Note that this refers to the actual number of lines as displayed on the screen, and not to the text formatting in the original document.

Furthermore, the HTML parser is by far not finished, and its author has discontinued its development. During testing, several issues were encountered, from which only the most critical part has been resolved (E.g. image resizing or redrawing the document after orientation change of mobile screen).

The list of current issues found in the HTML parser (most probably not complete, since the parser requires more thorough testing):

1. <title> tag not ignored
2. <!– comments –> not ignored if there are tags within them
3. height and width parameters of img tag cause problems
4. <h1> tag is being ignored, as probably all other heading tags
5. <span> tag is being ignored
6. justify align in <p> tag does not seem to work sometimes
7. end of <!– comment –> not recognised correctly, need an extra '=' character to break
8. <script> tag not ignored, when there is a <br> tag few lines above it
9. align sets align for all consecutive tags, not just the one it is a property of
10. <style> tag not ignored, if there are key characters in its content
11. <head> content should be ignored all together

From a quick look at the above list, it is obvious in what state the parser is, and how much work has to be done on it to make it reliable. It is recommended to only use the supported tags for the time being.

4.4.1.2 Deploying with a model

If a model is connected, or more specifically, if an FSharp.core library is present in the codebase libraries, the build process for WinRT platforms (Windows Store and Windows Phone) will fail. This is a documented and reported bug of Unity environment[16], and it

[16] Issue ID 674666 on Unity issue tracker (issuetracker.unity3d.com)

should be fixed soon after the finalisation of this thesis.

The attached model also causes problems when building the WebPlayer, because in the example application, there is a reference for library WindowsBase, which, for unknown reasons, can not be attached to the finished player.

A similar problem occurs when building for WebGL, but given the fact that in the time of the framework testing, this feature was still presented as experimental, there is a reason to expect that this problem is not caused by the project itself, but by some bug in the WebGL build processor.

4.4.1.3 Other

A completely different problem, that is not blocking the development itself, but can be annoying, is a 'ghost object' that is present in the scene. This object is initialised 4 times every time the game mode is activated, and shows warning level message in console that says 'The referenced script on this Behaviour is missing!'. This has also been reported as a bug, but since it is not in any way critical, tracking of the issue is not currently in place.

4.4.2 Build settings

Before building, it is recommended to go through the Player settings for the target platform. Here, the user can specify the name of the created application, as well as icons and many other settings.

Notable settings for Android and iOS devices are the bundle version and identifier, which should be used to correctly specify the application ID and version of the application. If the default value is left, the target device will not be able to distinct multiple different applications created in the RadApp framework, and the installation will override the application that is already on the device.

Other settings should be by default in such a state that allows successful deployment on the supported platforms.

4.4.3 Tested platforms

The deployment of the attached example application has been successfully tested on the following platforms:

- Windows 8.1 Standalone, x86
- Android 5.0.1 'Lollipop', on device Samsung Galaxy S4

The build process has also been successful for following platforms, but without a testing device for final deployment.

- iOS
- Mac OS X Standalone

- Linux Standalone

When the application does not have any model attached, hence the FSharp.core library can be removed, all of the other major platforms seem to work correctly. This includes the WinRT platforms and WebPlayer. Other platforms have not been tested, but as there are no extra libraries included, and the HTML parser is fully written in C#, and works flawlessly in the Unity Player, there should not be any particular reason for the application to not work on them.

4.5 Example application

For testing and demonstration purposes, an example application has been developed in the RadApp framework and released for Windows Standalone and Android. This application has a Modelica model, and is a re-implementation of the same functionality created in Silverlight, as part of the Bodylight standard workflow. This application, namely page 1 of this application, has been completely remade in the RadApp environment.

This application shows an interactive visualisation of an acid-basic concentration model, has questions and tasks and explicatory text attached. The release name of this application is AcidBaseApp.

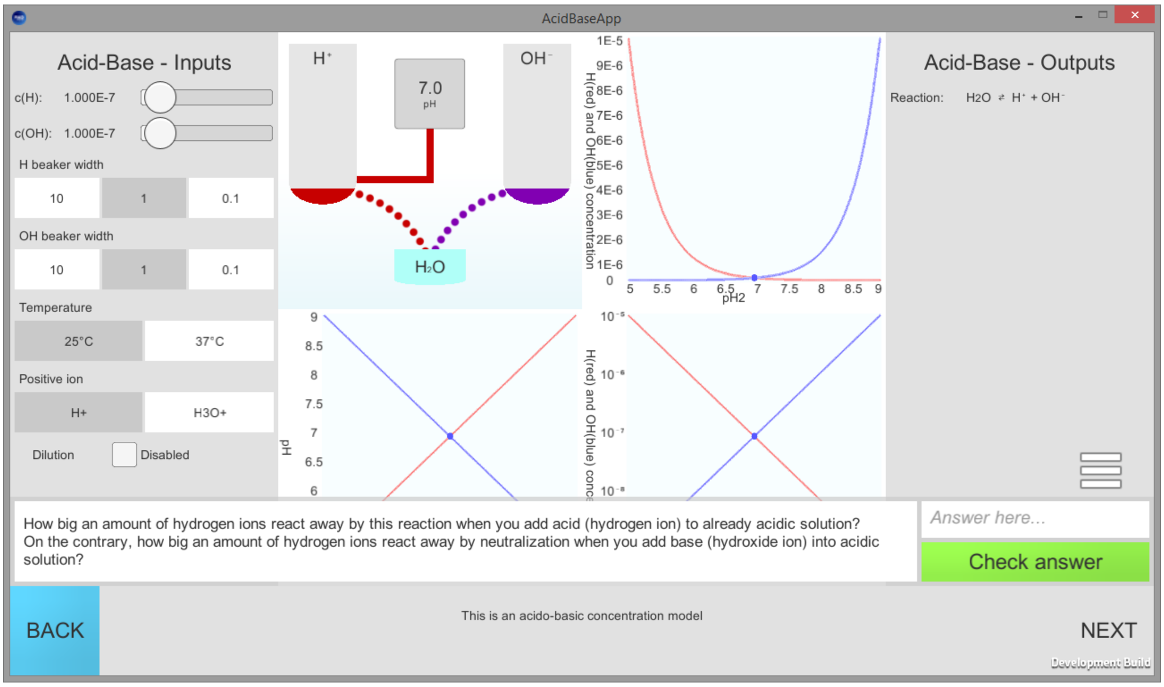
The release notes for this application can be found in attachment A. Here, the differences between the RadApp implementation, and the original application in Silverlight, is described.

4.5.1 Silverlight comparison

For visual comparison, see pictures 4.9 and 4.10, where you can see the RadApp implementation, and the original Silverlight page 1 interface, respectively. As clearly visible from the pictures, the structure of the visual content has been maintained. The only functionality that is not reproduced in the RadApp environment are one-way sliders, that allow only addition to each beaker. This is handled by the trigger labelled 'dilution', which, when disabled, should only allow moving the slider to the right. This would require overloading the standard Unity slider, and checking the user input for correct direction. There was some effort made to implement this functionality, but it remains unfinished.

The greatest difference is in the presentation of the texts. In the RadApp implementation, there are two screens preceding the model visualisations (see pictures in figure 4.11). On the first screen, the explicatory text is presented, on the second screen, the questions and tasks follow. Then, when the main content is displayed, the first question is again shown in the command bar. After answering it, the student is redirected to visually same content, but with the next question to solve. This way, the student actually has to interact with the application to prove that he has found the answer for the presented question, as opposed to the silverlight UI, where all the content is maintained in a single screen, without an interactive quiz.

On the following picture (4.12), the functionality specific to mobile devices is presented. The following screenshot has been taken in the landscape mode. It shows the first application



c(H): 1.000E-7

c(OH): 1.000E-7

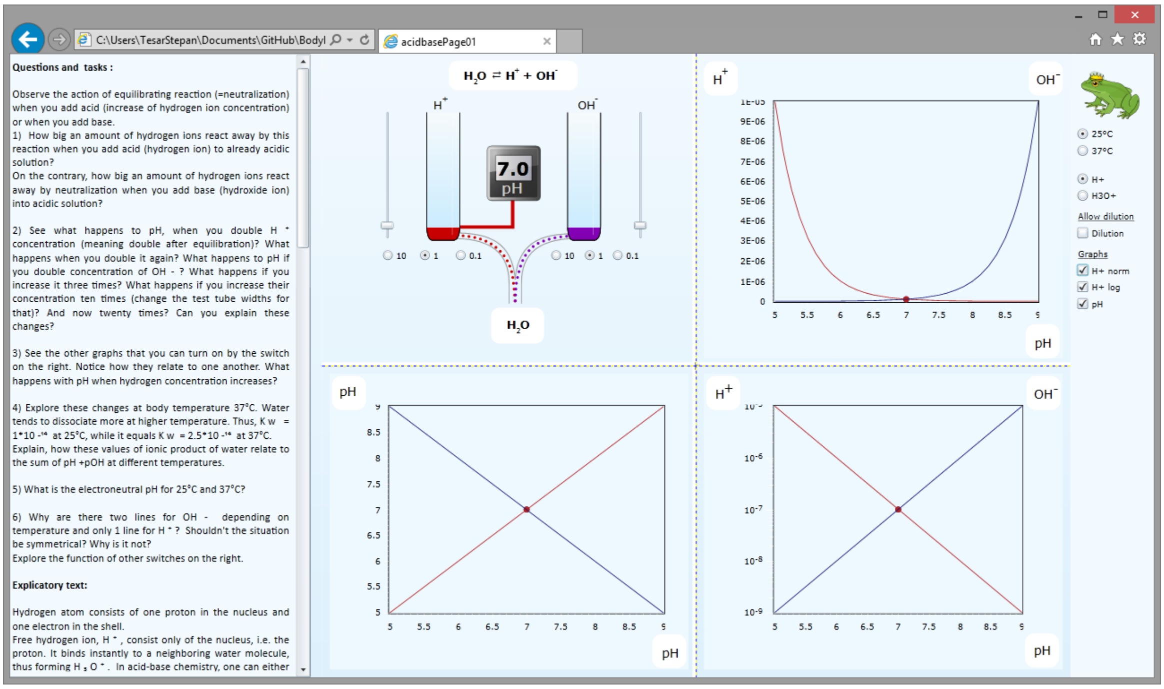
Dilution □ Disabled

Reaction: H2O ⇌ H⁺ + OH⁻

How big an amount of hydrogen ions react away by this reaction when you add acid (hydrogen ion) to already acidic solution? On the contrary, how big an amount of hydrogen ions react away by neutralization when you add base (hydroxide ion) into acidic solution?

This is an acido-basic concentration model

Figure 4.9: The example application AcidBase app, content with a connected model. The presented question is not well composed, as it asks two questions, but in this example application all text inputs are set to accept answer '1', as the application is done purely for functionality demonstration



● 25°C
○ 37°C

● H+
○ H3O+

Allow dilution
□ Dilution

Graphs
☑ H+ norm
☑ H+ log
☑ pH

Questions and tasks :

Observe the action of equilibrating reaction (=neutralization) when you add acid (increase of hydrogen ion concentration) or when you add base.
1) How big an amount of hydrogen ions react away by this reaction when you add acid (hydrogen ion) to already acidic solution?
On the contrary, how big an amount of hydrogen ions react away by neutralization when you add base (hydroxide ion) into acidic solution?

2) See what happens to pH, when you double H ⁺ concentration (meaning double after equilibration)? What happens when you double it again? What happens to pH if you double concentration of OH - ? What happens if you increase it three times? What happens if you increase their concentration ten times (change the test tube widths for that)? And now twenty times? Can you explain these changes?

3) See the other graphs that you can turn on by the switch on the right. Notice how they relate to one another. What happens with pH when hydrogen concentration increases?

4) Explore these changes at body temperature 37°C. Water tends to dissociate more at higher temperature. Thus, K w = 1*10 -¹⁴ at 25°C, while it equals K w = 2.5*10 -¹⁴ at 37°C. Explain, how these values of ionic product of water relate to the sum of pH +pOH at different temperatures.

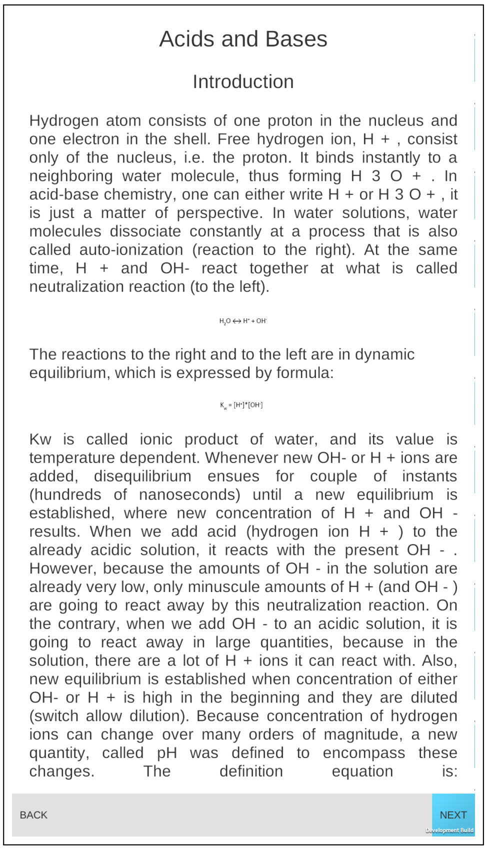
5) What is the electroneutral pH for 25°C and 37°C?

6) Why are there two lines for OH - depending on temperature and only 1 line for H ⁺ ? Shouldn't the situation be symmetrical? Why is it not?
Explore the function of other switches on the right.

Explicatory text:

Hydrogen atom consists of one proton in the nucleus and one electron in the shell.
Free hydrogen ion, H ⁺ , consist only of the nucleus, i.e. the proton. It binds instantly to a neighboring water molecule, thus forming H ₃ O ⁺ . In acid-base chemistry, one can either

Figure 4.10: The original AcidBase page 1. The basis for developing the AcidBase application in the RadApp framework.



Acids and Bases

Introduction

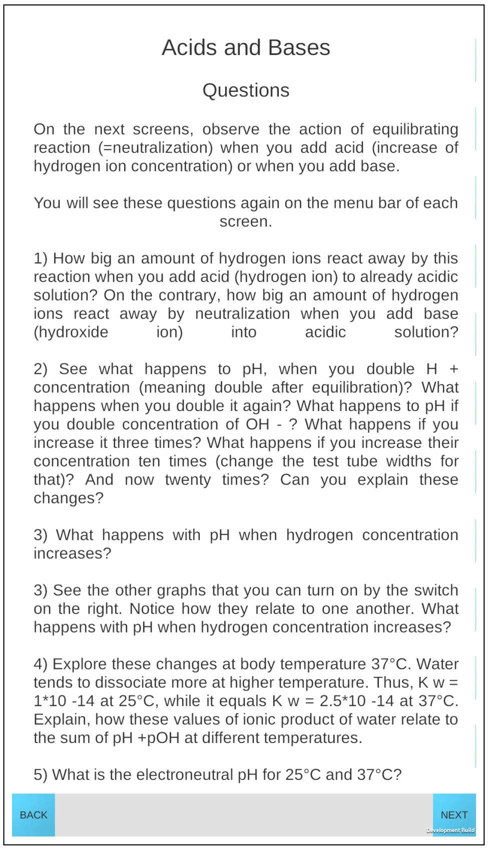
Hydrogen atom consists of one proton in the nucleus and one electron in the shell. Free hydrogen ion, H + , consist only of the nucleus, i.e. the proton. It binds instantly to a neighboring water molecule, thus forming H 3 O + . In acid-base chemistry, one can either write H + or H 3 O + , it is just a matter of perspective. In water solutions, water molecules dissociate constantly at a process that is also called auto-ionization (reaction to the right). At the same time, H + and OH- react together at what is called neutralization reaction (to the left).

$$H_2O \leftrightarrow H^+ + OH^-$$

The reactions to the right and to the left are in dynamic equilibrium, which is expressed by formula:

$$K_w = [H^+]*[OH^-]$$

Kw is called ionic product of water, and its value is temperature dependent. Whenever new OH- or H + ions are added, disequilibrium ensues for couple of instants (hundreds of nanoseconds) until a new equilibrium is established, where new concentration of H + and OH - results. When we add acid (hydrogen ion H +) to the already acidic solution, it reacts with the present OH - . However, because the amounts of OH - in the solution are already very low, only minuscule amounts of H + (and OH -) are going to react away by this neutralization reaction. On the contrary, when we add OH - to an acidic solution, it is going to react away in large quantities, because in the solution, there are a lot of H + ions it can react with. Also, new equilibrium is established when concentration of either OH- or H + is high in the beginning and they are diluted (switch allow dilution). Because concentration of hydrogen ions can change over many orders of magnitude, a new quantity, called pH was defined to encompass these changes. The definition equation is:



Acids and Bases

Questions

On the next screens, observe the action of equilibrating reaction (=neutralization) when you add acid (increase of hydrogen ion concentration) or when you add base.

You will see these questions again on the menu bar of each screen.

1) How big an amount of hydrogen ions react away by this reaction when you add acid (hydrogen ion) to already acidic solution? On the contrary, how big an amount of hydrogen ions react away by neutralization when you add base (hydroxide ion) into acidic solution?

2) See what happens to pH, when you double H + concentration (meaning double after equilibration)? What happens when you double it again? What happens to pH if you double concentration of OH - ? What happens if you increase it three times? What happens if you increase their concentration ten times (change the test tube widths for that)? And now twenty times? Can you explain these changes?

3) What happens with pH when hydrogen concentration increases?

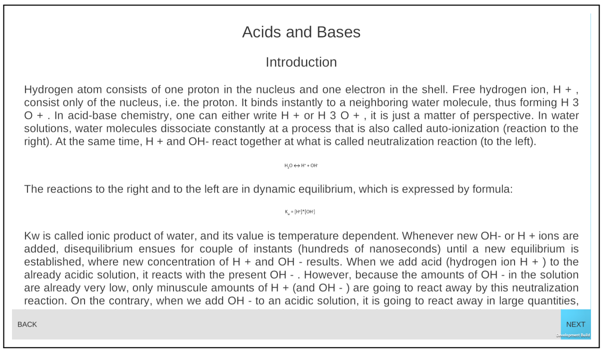
3) See the other graphs that you can turn on by the switch on the right. Notice how they relate to one another. What happens with pH when hydrogen concentration increases?

4) Explore these changes at body temperature 37°C. Water tends to dissociate more at higher temperature. Thus, K w = 1*10 -14 at 25°C, while it equals K w = 2.5*10 -14 at 37°C. Explain, how these values of ionic product of water relate to the sum of pH +pOH at different temperatures.

5) What is the electroneutral pH for 25°C and 37°C?

Figure 4.11: First two screens (from left to right, respectively) in the mobile version of AcidBaseApp, shown in the portrait mode.

screen again (the same content as on the left picture in figure 4.11), but automatically fitted into the screen after the device was physically rotated.



Acids and Bases

Introduction

Hydrogen atom consists of one proton in the nucleus and one electron in the shell. Free hydrogen ion, H + , consist only of the nucleus, i.e. the proton. It binds instantly to a neighboring water molecule, thus forming H 3 O + . In acid-base chemistry, one can either write H + or H 3 O + , it is just a matter of perspective. In water solutions, water molecules dissociate constantly at a process that is also called auto-ionization (reaction to the right). At the same time, H + and OH- react together at what is called neutralization reaction (to the left).

$H_2O \leftrightarrow H^+ + OH^-$

The reactions to the right and to the left are in dynamic equilibrium, which is expressed by formula:

$K_w = [H^+]*[OH^-]$

Kw is called ionic product of water, and its value is temperature dependent. Whenever new OH- or H + ions are added, disequilibrium ensues for couple of instants (hundreds of nanoseconds) until a new equilibrium is established, where new concentration of H + and OH - results. When we add acid (hydrogen ion H +) to the already acidic solution, it reacts with the present OH - . However, because the amounts of OH - in the solution are already very low, only minuscule amounts of H + (and OH -) are going to react away by this neutralization reaction. On the contrary, when we add OH - to an acidic solution, it is going to react away in large quantities,

Figure 4.12: The first screen in the mobile version of AcidBaseApp, shown in the landscape mode.

Graphs are the most notable advantage of the Silverlight version, since they provide much better functionality, are scalable (the cross separating them can be moved), and even their implementation in the code provides better functionality for the programmer. In RadApp, own, simple implementation of similar graphs had to be done, since all the Unity plugins offering advanced graph functionality are usually both paid and require Unity professional version.

As Silverlight is not currently a very prospective platform, a big advantage of RadApp is its multi-platformity and promising development environment. Since Bodylight can also easily target WinRT, which RadApp is currently unable to do, due to a problem in Unity (see section 4.4.1), this again talks in favour of the original Bodylight UI.

Regarding the implementation side of this topic, it is undoubtedly more clean to use the original Bodylight process of UI design, since the framework is optimised for this workflow as a whole. But RadApp does provide several advantages and benefits. RadApp also has different targets than Bodylight framework, but their interconnection certainly brings benefits. To RadApp, by providing useful features, and to Bodylight, by providing insight into a new form of connecting a user interface.

Chapter 5

Discussion

The product of this work is a complete framework for creating e-learning applications. There is no behaviour that would target the medical fields specifically, but by providing methods for Modelica model connection, we present wast options of highly specialised applications, including applications for medical fields.

5.1 Attained goals

As a review of the work, original goals were compared with the finished product. Firstly, lets reiterate the original requirements of capabilities that the framework should provide in the application it was used to develop:

- Displaying text, image, videos and Interactive animations, that can also be combined.
- Application provides separate screens and allows data transfer between them.
- The screens can be interlinked into a graph-like, non-linear structure.
- User can easily switch between the screens and interact with the content via a unified interface.

There were also requirements for the framework itself. These include platform independence, scalability, permissive licence, low financial cost, interconnectivity with existing frameworks - namely Bodylight. And of course, the ultimate main objective was to create a framework that will reduce time required for the application development

Each item on the previous list is to be evaluated with the results presented in the chapter 4 - Realisation.

5.1.1 Content types

As shown on pictures in figures 4.11 and 4.12, the application can correctly display longer texts, with embeded pictures (unfortunately, the pictures - containing formulas - are not well visible on the mobile version of the application, since it is not optimised for use on smartphones). This not only demonstrates the ability of the application to display texts and pictures, but also the ability to show them together on a single screen, which covers the content combination requirement.

Furthermore, on picture 4.9, an animated beaker setup is displayed. The beakers smoothly change their levels to the values shown in the left panel, represented also by sliders, and during the change, the points leading from the H or OH beaker to the water containing beaker move with according speed.

The only type of content that is currently not possible to present, are videos. This problem has already been described in section 4.2.11.1, and since its workaround would be extremely time-consuming, while it can be simply overcame by upgrading to the professional version of Unity, it has not been solved in the current version of the framework. Nevertheless, the solution for this problem exists, and is described in this thesis in the previously referenced section.

5.1.2 Screens and content structure

Visible again from pictures of the example application (4.11, 4.12, 4.9), there are multiple screens, each displaying different content. User can navigate between the screens via the ’next’ and ’back’ buttons. It is also possible to reach away from this linear structure by using buttons in the action bar (see picture 4.6), which can lead to multiple different contents, including contents used already in the linear structure. Allowing branching and loops in the structure ultimately leads to a structure that can be described as a graph.

It is also possible to pass arguments from one screen to another, using the ’target state’ parameter, which can be set for all buttons, whose function is to redirect the application to another screen. The architecture allowing this behaviour is described in section 3.2.2, and its usage in section4.2.4, or for the action bar content in section 4.2.5.

5.1.3 Unified interface

The graphical user interface was designed in such a way, that each screen contains the same elements. Basic screen, without any interaction, only contains the navigation bar, and can optionally also show action bar with question forms of predefined structure, or furthermore parameter bars, if the content has an attached model. This layout and its behaviour is described in section 3.3.1. The important fact is, that the design remains the same throughout not only the application screens, but also all the applications developed using the RadApp framework.

5.1.4 Platform independence and scalability

Presented in section 4.4.3 is a list of the platforms for which the application build process was successful. While the application deployment was only tested on Windows 8.1 and Android, the application build for Mac OSX or Linux produces a final, deployable package, and since there was no error output during the build process, it is possible to assume the correct functionality on these platforms as well.

The previous paragraph also only accounts platforms available for applications that use full potential of the RadApp framework, and thus include a Modelica model. Stated again in the same section (4.4.3) is, that without a connected Modelica model, there is no need to include any external libraries, and all code is therefore managed directly by Unity. Therefore, the application can technically be deployed on any platform that Unity provides. Still, only deployment for web-player and WinRT (Widnows Store and Windows Phone) has been tested.

On all tested devices the GUI has appropriately scaled all its components. The text on mobile devices is automatically scaled to be easily readable, as described in section 4.2.9. The interface also adapts on devices with screen orientation, which can be seen on pictures 4.11 and 4.12.

5.1.5 Open licence and low cost

The development has stayed financially cost-free until the finalisation of this thesis. This has been one of the major requirements, and it has been fulfilled even with a cost of sacrificing other features that were originally required (such as video, as described in the previous sections (5.1.1). This has also led to selection of tools that are published for free. Such tools usually also come as open-source, as since the author does not have any monetary profit from the tool, there is no reason for him to not allow modifications of his work by the community.

The only part of the system that is explicitly licensed, is the html parser (described in section 4.4.1.1), which has been released under the MIT licence. This permissive licence does not prevent any usage or modifications, but requires the user of the product to give credit to the author (Ruslan A. Abdrashitov), and to release the final product under the same licence. The related code is marked in such a way, that makes it obvious who created it, thus keeping the licence conditions. Furthermore, the credit is also given in the text of this thesis on appropriate places (such as in this paragraph, or in section 4.4.1.1 where the parser is described). Since the framework has not yet been publicly released, as it currently serves mainly for this thesis purposes, licensing has not yet been an issue. As a remark for the future, the MIT licence is permissive, and thus releasing the framework under it would not hinder the original requirements.

5.1.6 Bodylight connectivity

Designed structure of the connection of Bodylight to RadApp in Unity is described in section 3.2.1. Description of the successful implementation of this design, and consequently fulfilment of the requirement can be found in section 4.3. The proof of success is also demonstrated in the example application, specifically on picture 4.9, where visual components are controlling - and also are controller by - a model running in the background of the application. Keep in mind, that the same application is running on Windows and on Android, and therefore proving that the model is packaged within the application, and not running separately.

Although connecting the model creates certain limitations, particularly in the deployment options (see 4.4.1.2), the requirement of attaching a Modelica model is fulfilled, while still keeping the framework largely multi-platform.

5.1.7 Overall competition

There are several unfinished issues, that cover smaller tasks, or cover unimplemented feature that would move the framework further. Out of 137 current issues (that contain not only bugs, but also tasks or feature propositions), 3 are incomplete, and 11 are in an unresolved state (those include some of the problems mentioned in previous paragraphs). Out of these unfinished 14, only 3 are actual faults in the project, or the example application. The rest are propositions for future work (as described further in 5.1.9), or improvement ideas, that were not implemented, or even discussed upon (so it is not always clear, if the feature would be useful and eventually included).

The development itself, without including the previous research or familiarisation with the used tools, begun in min-December, 2014, and practically has not ended yet. In the state used for the advocacy of this thesis, the project has been conserved in the beginning of May, 2015. This means that the development took 4,5 months of practically daily work. Excluding the weekends, and averaging the workday to 6 hours, gives a result of a little over 700 work hours. Admittedly this number also includes time spent on composition of the thesis text, which consumed a more that insignificant part of the time. Considering the scope of the development, the limited time and the relatively small amount of remaining work, this can be deemed a good result.

5.1.8 Development time reduction

Although an empiric study of the attained time reduction was not performed, specific parts of application development process are already implemented and prepared in our framework. These include the application structure and logic (summarised in 5.1.2), the user interface (5.1.3) and optimisation for multiple platforms (5.1.4). Furthermore, tools for creating the content in a simple and visual fashion are provided (examples of such tools viz 4.2.2, 4.2.4, or 4.2.5), as well as a tested workflow and prepared interface for a Modelica model connection (described in 4.3.2).

By providing a majority of the finished product, the time spend on creating the rest will logically be reduced. This has been partially tested when creating the example AcidBase application, which only took days to complete (as opposed to weeks, or even more likely months, usually spent with a conventional development approach). At the time of AcidBase development, the framework was still largely unfinished an untested, an so a development of similar application using the current state of the framework would be even more reduced, possibly to only one or two days.

The only additional time required is spent while familiarising with the provided tools, which is simplified by the User Manual (provided in attachment B). By taking the scope of provided implementation into account, it can be confidently stated, that time spent becoming familiar with the provided framework would not exceed the time otherwise spent on the additional implementation. It is even possible to create simple applications without any programming knowledge. In this case, the saved amount of time (or money, in cases when a professional would have to be hired for the creation of the application) is evident.

5.1.9 Future work

As discussed previously in this chapter, or even in section 4.4.1 in previous chapter, several problems or minor features remained unfinished, because they had to be pushed back in the priority list, or the problems cloaking their development became too overwhelming.

Video content would an extremely useful feature to have. In the Unity professional, this is a very simple task as well, and for this particular reason, there was no motivation to spend time or trying to implement this functionality in other way. One attempt was the use of Awesomium plugin (see section 4.2.7 or 4.2.11), but since this branch of development has been discontinued, there were no future efforts to implement video content.

WinRT deployment is a matter of patience. When Unity fixes the FSharp support for this platform, there might still be further issues involved, before the deployment is possible, but until then there is no way of advancing the work in this matter any further.

One way sliders that were described in section 4.5.1. There is a partially implemented interface in the model interval value and the connected visual components, but as it started to consume unreasonable amounts of time to debug this feature, it was scratched, and only the interface methods remain.

Graphs can be either completely rewritten, or require greater amount of work to match the functionality of other conventional graph systems. Much easier solution is to replace the current graphs with some commercial solution, but with the target of maintaining the development cost-free, this was not possible within this thesis. And expanding the graph functionality beyond the necessary would require great amount of time, which had to be spent on more important tasks in hand.

Html parser provides wast possibilities of improvement. Firstly, it has to go through an in-depth testing, and all the found problems with currently supported tags have to be resolved (see section 4.4.1). Then, it should be expanded to detect all html tags, and to handle them correctly. The list of currently known issues with the parser is listed in section 4.4.1. After resolving all the current problems, css support can be added, to truly support visual styling. This alone would elevate our framework by a whole level, but the task would be so great, that is would again be simpler to use one of the commercial solutions for displaying live web pages.

Minor tasks. Throughout the development, many less important tasks were postponed, and still remain in the issue list. These do not prevent the correct functionality, but they might be a nice-to-have features, or on the contrary, some minor problems that can be annoying either to the developer, or the user of the final application. Follows a list of example issues that remain unfinished:

- Application demands ridiculous permissions for installation (network access)

- Extend the generic dictionary to automatically use variable name
- Dispose of distant contents to save memory
- Disable auto-rotation of screen if it is disabled on device

Without further description, the list only demonstrates the nature of issues that remain to be solved.

5.2 Conclusion

Comparison of the realised framework with the requirements that were set up at the beginning shows, that all the major objectives were attained. With a partial exception of included content types, which is a missing interface for displaying in-application videos. For this unresolved issue, a solution is still presented, which was only not incorporated into the project in order to accomplish a more important goal: keeping the development financially and legally unbound. More importantly, by creating the example application, it was shown, that the development time necessary for creating a multi-platform application using the techniques provided by the created framework, is significantly shorter than creating an application from scratch, by saving time on many levels of development. Thus, the proud conclusion is that the work has been successful.

Bibliography

[1] Pekka Abrahamsson. *Agile Software Development Methods: Review and Analysis*. VTT publications, 2002.

[2] Tim Berners-Lee, Robert Cailliau, Ari Luotonen, Henrik Frystyk Nielsen, and Arthur Secret. The world wide web. *Communications of the ACM*, pages 37(8):907–912, August 1994.

[3] Robert C. Miller and Brad A. Myers. Lightweight structured text processing. *Proceedings of the USENIX Annual Technical Conference*, June 1999.

[4] Jonas Echterhoff. On the future of web publishing in unity, April 2014. blogs.unity3d.com [Online; posted 29-April-2014].

[5] Justin Ferriman. Top 5 elearning development programs, March 2014. www.learndash.com [Online; posted 23-March-2014].

[6] Bill Gardner. Current logo trends, 2014. logolounge.com [Online; posted 9-April-2014].

[7] O. C. Haas and K. J. Burnham. Systems modeling and control applied to medicine. *OC Haas, KJ Burnham, Intelligent and Adaptive Systems in Medicine*, pages 17–52, March 2008.

[8] Arne Hildebrand, Thomas C Schmidt, and Michael Engelhardt. Mobile elearning content on demand. *International Journal of Computing & Information Sciences*, 5(2), 2007.

[9] Simon Hill. Android 5 vs. ios 8 vs. windows phone 8.1: Which smartphone os is best?, 2014. digitaltrends.com [Online; posted 27-July-2014].

[10] Sarah Horton. *Access by Design: A Guide to Universal Usability for Web Designers*. New Riders, July 2005.

[11] Filip Ježek, Martin Tribula, Jiří Kofránek, Josef Kolman, Pavol Privitzer, and Jan Šilar. Set of educational interactive simulators - the results of bodylight framework. *Sborník příspěvků MEDSOFT 2013*, pages 38–48, 2013.

[12] Ray Jimenez. Rapid e-learning design and development: Part 1, October 2005. learningsolutionsmag.com [Online; posted 3-October-2005].

[13] Maurice Kindermann. Ux comparison chart, 2013. kintek.com.au [Online; posted 14-June-2013].

[14] Jiří Kofránek, Michal Andrlík, Tomáš Kripner, and Petr Stodůlka. From art to industry: Development of biomedical simulators. *Special Issue on the Research with Elements of Multidisciplinary, Interdisciplinary, and Transdisciplinary: The Best Paper Selection for 2005*, pages 62–67, 2005.

[15] Jiří Kofránek, Marek Mateják, and Pavol Privitzer. Hummod - large scale physiological models in modelica. *Proceedings of 8th. International Modelica conference 2011*, 2011.

[16] Jiří Kofránek, Pavol Privitzer, and Marek Mateják. Web simulator creation technology. *MEFANET report 03*, pages 32–97, 2010.

[17] Aarthy Krishnamurthy and Rory V O'Connor. An analysis of the software development processes of open source e-learning systems. In *Systems, Software and Services Process Improvement*, pages 60–71. Springer, 2013.

[18] Mark Masters. Unity, source 2, unreal engine 4, or cryengine - which game engine should i choose?, March 2015. blog.digitaltutors.com [Online; posted 5-March-2015].

[19] Xiufeng Pang, Raj Dye, Thierry Stephane Nouidui, Michael Wetter, and Joseph J. Deringer. Linking interactive modelica simulations to html5 using the functional mockup interface for the learnhpb platform. *Proc. of the 13th IBPSA Conference*, pages 2823–2829, August 2013.

[20] John Ricitiello. Unity 5 launch, March 2015. blogs.unity3d.com [Online; posted 3-March-2015].

[21] Clarence Simpson. Using unity for non-game app development, November 2014. www.jarcas.com [Online; posted 18-November-2014].

[22] A.W. (Tony) Bates. *Technology, E-learning and Distance Education.* Routledge, second edition, August 2005.

[23] Eric Von Hippel. Learning from open-source software. *MIT Sloan management review*, 42(4):82–86, 2001.

[24] Håkon Wium Lie and Janne Saarela. Multi-purpose publishing using html, xml, and css, May 1998. http://www.w3.org/ [Online; posted 7-May-1998].

[25] Brandon Wu. Top 10 reasons to choose unity 3d for app and game development, November 2013. www.pepwuper.com [Online; posted 19-November-2013].

Appendix A

AcidBase release notes

AcidBase 0.3.1 Release Notes

Bc. Štěpán Tesař

April 30, 2015

Contents

1 Introduction

The AcidBase application has been created as an example usage of the RadApp framework with a connected Modelica model. It introduces the principles of acido-basic equilibrium in an explicatory text, and then displays an interactive animation and graphs, that serve to demonstrate the principles. There are several questions presented. As the application is purely for demonstration of RadApp capabilities, and not intended for actual distribution for tutoring, the accepted answers are currently all set to '1'(without quotes).

1.1 Requirements

The application is released as a Windows standalone executable, and as an Android app. There are currently no known prerequisites.

The application is developed in the Unity game engine, which specifies the following minimal requirements (we provide the whole list, even though the application has not been released for some of the platform)

- Desktop
  - OS: Windows XP+, Mac OS X 10.7+, Ubuntu 10.10+, SteamOS
  - Graphics card: DX9 (shader model 2.0) capabilities; generally everything made since 2004 should work.
  - CPU: SSE2 instruction set support.
  - Web player supports IE, Chrome, Firefox, Safari and others.
- iOS: version 6.0 or later.
- Android: version 2.3.1 or later, ARMv7 (Cortex) CPU or Atom CPU, OpenGL ES 2.0 or later.
- Blackberry: version 10 or later.[1]

1.2 Tested platforms

Deployment of the application has been tested on following platforms:

- Windows 8.1 Pro x64
- Android 4.4 'KitKat' on Samsung Galaxy S4
- Android 5.0.1 'Lollipop' on Samsung Galaxy S4

[1] List adapted from unity3d.com

2 Instalation

2.1 Windows

The application has no self-extracting installer. The package can be placed anywhere and run directly. Only make sure that the AcidBaseApp_Data folder is at the same level as the main executable file.

Removing the application is as simple as deleting all the provided files.

2.2 Android

The application is not distributed through the Android App Store. It is therefore necessary to transfer the apk file to the phone storage (E.g. via USB cable). The option in 'Settings'->'More' (or 'Other')->'Security'->'Allow installation from unknown sources' has to be enabled. Afterwards, find the apk file using custom file manager, and click it to start the installation process. The application icon will automatically appear in the 'Apps' menu when the installation is successful.

Uninstalling the application is same as any other application on the Android device, which depends on the version of the OS. Standard procedure is to go to 'Settings'->'More' (or 'Other')->'Application Manager' where several tabs are present. The AcidBase application should be present in either the 'Downloaded' or 'All' section. After finding it, click the row to open the app info, where an option to uninstall the app is present.

3 Usage

After starting the application, a short Unity splash screen will be displayed. In a short time, the first application content, 'Introduction', will be displayed. By swiping the touch-screen with one finger from right to left, or pressing the 'Next' button in the bottom right corner, you will be shown the next content 'Questions'. The following content is the acido-basic model visualisation. The screen on devices with screen rotation enabled will automatically rotate to the 'landscape' mode.

At the right part of the screen, the controls of the model are displayed. In the middle, the model visualisation and graphs, and to the right, the model outputs are displayed. By changing the values using the controls, you can observe changes in the visualisation and graphs.

Answering the presented question at the bottom of the screen (currently the accepted answer for all questions is '1'), you can get through all the questions all the way to the final screen. By pressing the 'Back' button in the bottom left corner (or swiping the screen from left to right, or pressing the 'go back' Android button), you will display the previous content.

Closing the application can be done by clicking the 'Close' button. On Windows, this is the 'X' labelled button at the top right corner of the application window. On Android, this the standard middle button, which can be either a hardware, or conditionally displayed software button at the bottom of the screen.

4 Known problems

As the application has been developed using RadApp version 0.3.1, which as the version number suggests, is not a finished, market-ready, product. For this reason, there are several known issues. These should not prevent the general usage of the application (the application is not known to 'crash').

4.1 Internet access

On both platforms, the application seems to request internet access. On Android, this is seen before the installation as an item in the 'Application access' list. On windows, a firewall permission will be requested. The firewall permission can be denied, since the application does not actually require any internet connectivity, and will not use any bandwidth even on Android devices.

4.2 Graph behaviour

The graphs in the model visualisation should redraw appropriate lines, when the 'Temperature' parameter is changed, but this behaviour acts correctly only after switching the temperature 2 times.

On some resolutions, the axes labels might be incorrectly aligned to the actual graph scale. This is due to imperfect implementation of the graph UI. The correctly displayed labels should align the lowest values with the bottom left border of the graph, and the greatest values with the top/right end of the axes, and distribute the labels in between them evenly.

4.3 Dilution

The dilution switch currently has no effect on the model, as it should only serve to prevent the c(H) and c(OH) sliders to set a value smaller than the one currently set, but this functionality has not been implemented yet.

4.4 Beaker levels

Only applies to Android.

In the model visualisation, the beaker should display a 'liquid' with a level that corresponds to the concentration ratio of the particular ion. The graphics displaying the 'liquid' seems to be buggy on the tested Android device, which is believed to be due to the used rendering method and screen type.

4.5 Html content

Only applies to Android.

On certain devices, one or both of the first application screens (introduction and questions) can be displayed as black rectangle. This is due to the technique of html rendering and the limited texture size, that can be displayed on mobile devices. The current version is running without this issue on the previously listed devices (at a price of missing last 3 images in text on the first screen), but since the specifications of Android device vary wildly, this issue can occur on other devices, presumably mostly on older types of phones.

4.6 Connected usb device

Only applies to Windows (probably).

When a usb cable with a multimedia device is connected while the AcidBase application is running, a box in bottom left corner is displayed, with an error message "<RI.Hid> Failed to get device caps ...". This message can repeat itself in the box along with other messages depending on the device type.

This is because Unity automatically detects multimedia devices and tries to retrieve some information. This only occurs because the current build is set as 'Development build' (which is displayed in the bottom right corner), and the error box serves for testing purposes. It can be hidden by the 'close' button next to the box. This issue has not been observed on Android devices.

Appendix B

RadApp user manual

RadApp 0.3 User Manual

Bc. Štěpán Tesař

May 1, 2015

Contents

Chapter 1

Introduction

This guide should provide information about how to set up and use the RadApp framework, including the necessary software and optional tools.

The described practices apply to RadApp version 0.3.x (x being mandatory number), but the general architecture of the framework should not change too dramatically in the near future. Therefore, this manual should be applicable to further version of the framework, before the appropriate revision of this of this guide is released.

Chapter 2

Requirements

To start developing in the RadApp framework, it is necessary to install the required software first. In this chapter you will find a list of required and optional software, and a brief explanation of how to obtain each tool.

2.1 Unity

The main part of framework is a package for Unity game engine. This package includes necessary code that take care of the composition of the final application, and visual editors used to set up the application.

RadApp has been developed using Personal Edition of Unity 5.0.1, which can be freely used under conditions stated in the legal agreement of Unity, section 3, as follows:

2.1.1 Unity Personal Revenue Restrictions

Unity Personal (including the iOS and Android platform deployment options) may not be used by:

1. a Commercial Entity that has either: (a) reached annual gross revenues in excess of US$100,000, or (b) raised funds (including but not limited to crowdfunding) in excess of US$100,000, in each case during the most recently completed fiscal year;

2. a Non-Commercial Entity with a total annual budget in excess of US$100,000 (for the entire Non-Commercial Entity (not just a department)) for the most recently completed fiscal year; or

3. an individual (not acting on behalf of a Legal Entity) or a Sole Proprietor that has reached annual gross revenues in excess of US$100,000 from its use of the Software during the most recently completed fiscal year, which does not include any income earned by that individual which is unrelated to its use of the Software.[1]

There are further terms of use available online on unity3d.com, that the user should get familiar with before using the Unity software.

[1] From the Unity End User Licence Agreement document, available online at unity3d.com/legal/eula

2.1.2 System requirements

Unity has specified minimal system requirements for its use for development. These consequently apply to the RadApp framework as well:

Operating system

- Microsoft Windows XP (SP2) or 7 (SP1)[2] or newer.[3]
- Mac OS X 10.8 or newer[4]

GPU [5] Any GPU with DX9[6] (shader model 2.0) capabilities. This is practically any GPU developed after year 2004.

Others Further, we recommend at least 2GB or RAM and an Intel Core i5 processor or equivalent. Unity works well on both 32- and 64-bit systems.

2.1.3 Installation

Unity is a professional tool, that comes with a self-extracting installer that will guide you through the process, which can be obtained from the Unity web pages at unity3d.com.

2.2 Visual Studio

This tool is necessary if you wish to include a Modelica model in you application, or if you wish to create applications for Windows Store or Windows Phone.

It is recommended to use Visual Studio 2013 Professional, but all required capabilities are available in the free versions of Visual Studio Community or Express. These tools can be obtained from the Visual Studio web page at www.visualstudio.com.

2.3 Model

If you wish to include a Modelica Model in your application, you will need to implement certain parts within the Bodylight framework, which requires Visual Studio 2013 and .NET framework 3.5.

For the model implementation you can use any favourite tool, but it is required to have OpenModelica 1.9.1 installed for translation of the model to a C# file.

Bodylight is obtainable as a separate package with prepared interface for interconnection with the Unity RadApp project.

[2] SP stands for Service Pack

[3] Windows Vista is not supported, and server versions of Windows are not tested.

[4] Server versions of OS X are not tested.

[5] Graphics Processing Unit, also known as 'Graphics Card'

[6] Microsoft DirectX technology version 9

2.4 Additional

There are more tools that can be further used during the development. For example, should you wish to create 3D meshes to display a custom graphics, there are free tools that can be used.

Inkscape is free vector graphics editor, which can export the graphics to svg. Blender can then be used to convert the file to a 3D mesh, that can then be directly imported to Unity.

2.4.1 Specific platform development requirements

Developing applications for some platform also require additional tools. Unity will automatically search for some of these tools before the build process for the particular platform can start.

Android For Android applications, it is necessary to have Android Software Development Kit(SDK) installed. This kit can be downloaded from the android developer web page at developer.android.com. It is not necessary to install any additional editors, the SDK standalone package is sufficient. Unity will also automatically download any required components of the SDK before the first build.

Furthermore, it is necessary to have Java Development Kit installed. This might be required even before the installation of the Android SDK. This kit is available from the oracle web page at www.oracle.com, section 'Downloads', 'Java SE'. Be sure to download the correct package for your operating system and processor architecture.

Windows For Windows standalone, it is not necessary to have any further tools installed.

Windows Store/Phone apps can be built in Unity, but need to be further compiled in Visual Studio (see 2.2). The compilation for these platforms is only possible on Windows 8/8.1 operating system. The compilation will require additional tools, such as .NET 4.5 (which should be included in the Windows 8/8.1 implicitly) or specific packages, that Visual Studio should download automatically.

iOS iOS applications also have to be recompiled using Xcode 6.x. This is only possible on a Mac Computer, running OS X version 10.9.4 or newer. This build process has not been tested.

Blackberry Building for Blackberry requires 32 bit Java Runtime (JRE, not JDK). This build process has not been tested.

Chapter 3

Usage

In this chapter, is is described how to set up the framework and its usage, including description of the UI and its functions. A part dedicated to the Modelica model connection process is also included here.

3.1 Setting up the framework in Unity

After installing Unity, start it and when the Projects selection window opens, navigate to the RadApp package folder in the 'Open Other' dialog. This will open the RadApp project. Next time you run Unity, RadApp will be displayed in the Projects list.

After Unity has started and RadApp framework has loaded, you will see the basic Unity interface and RadApp application UI preview.

3.2 Unity interface

Although it might take some time to get used to the Unity interface, the RadApp project is done in such a way that should allow even people without deeper programming knowledge to be able to produce a simple application. There are 5 main components in the Unity editor you should get familiar with.

3.2.1 Project

This window displays file structure of the Unity project. RadApp has its files separated into a logical folder tree.

Codebase contains most of the scripts in Unity. To create a basic application, you will only need the ContentController file in this folder. There are also useful utilities, that can be used for development of more complex content (graphs, timers etc.) in the Utils subfolder, but these have prepared examples that have a more convenient interface implemented.

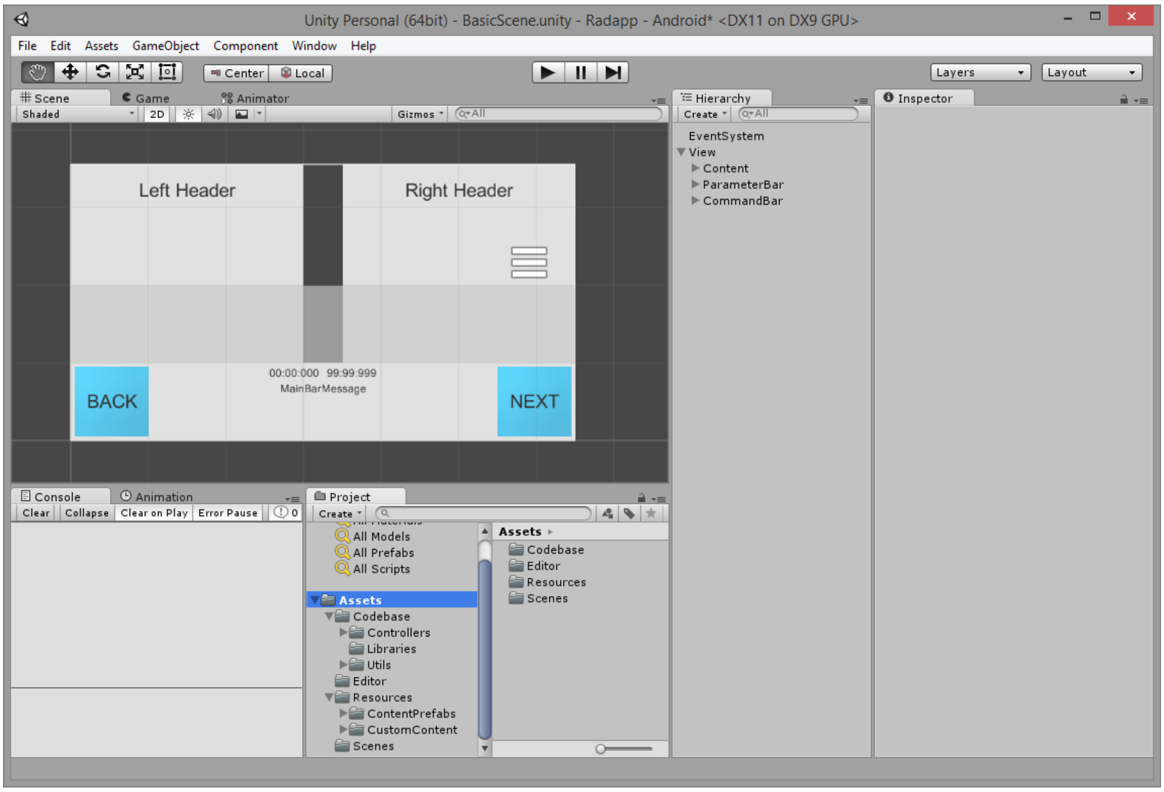


Figure 3.1: The interface of Unity after opening the RadApp project. Note that some of the window components can be relocated, or hidden. Look for the names of the tabs, and by dragging them you can relocate them to match the picture.

Editor folder contains custom inspector UI for some of the scripts. Content of this folder is not meant to be manipulated.

Resources is the folder where you will find useful components for creating the application (ContentPrefabs subfolder), and also where you should put all your custom made content (CustomContent subfolder).

The usage of particular components in this folder will be discussed further.

Scenes this is where Unity scene files are located. There is only one scene in RadApp framework, so this folder will serve no purpose when developing an application.

3.2.2 Hierarchy

might seem similar to the Project window at first, since it also displays a component tree. However, here, it does not represent a file structure, but rather the structure of components in the Application that you are creating. Each line in the hierarchy represents one 'GameObject', which is a basic unit in a Unity application. You can notice, that when you select a line in the hierarchy, a component in the 'Scene' window is also selected, and vice versa. Selecting a component will also display its attached scripts in the Inspector window, which will be discussed further.

The components in RadApp framework are grouped into several parts, corresponding with their purpose and visual representation in the application UI.

There are two basic components: 'EventSystem' and 'View'. You can ignore the 'EventSystem'. It only serves as the input control, and you wont need to manipulate it. 'View contains the UI components.

The most important for you, as the application designer, will be the subfolder 'Content'. Here, you will be placing the content you wish to display, as described in detail further in this chapter (3.3.1). The other two subfolders hold the application menu bars. You wont need to edit anything here, since all the settings of these bars are done when adding the content.

3.2.3 Inspector

is where the settings of every selected component are displayed. The meaning of these settings are again explained further.

3.2.4 Console

shows messages from the application during testing, and is priceless when debugging the application. If there is a problem during the application run, it is described here. Some errors can appear even during the composition of the UI, but these usually have no constructive meaning and can be ignored (such as ArgumentException: GUILayout: Mismatched LayoutGroup.Repaint).

3.2.5 Scene and Game

This is where you can see a preview of what you application will look like. in Scene, you can manipulate with the content (e.g. move and rotate the UI elements) and in the Game window, you can test your application to see how it performs when it will be deployed. To start the game mode, you have to press the 'Play' button at the top of the unity window, and press it again to exit it.

3.3 Setting up new application

By simply opening the RadApp project, you are set to start developing new application. If you wish to maintain a clear RadApp copy, you can duplicate the containing folder, and open the new copy. Once you are in the Unity editor, you can start adding you own content.

3.3.1 Adding content

Adding content to the application is simple. You navigate to the View/Content object in the Hierarchy window, and you create a new content here.

If you wish to add a html content, you can drag and drop a prefab named 'ContentHtml' from the Assets/Resources/ContentPrefabs folder in the project window, to the 'Content' component in the hierarchy window.

If you wish to create a new custom content, simply right click the 'Content' component, and select Create Empty. This will create a new child element under the Content component. To mark this object as a content, you have to attach the 'ContentController' script from Assets/Codebase folder in the Project. This is done by dragging the script file to the Inspector window, while your content is selected in the Hierarchy window.

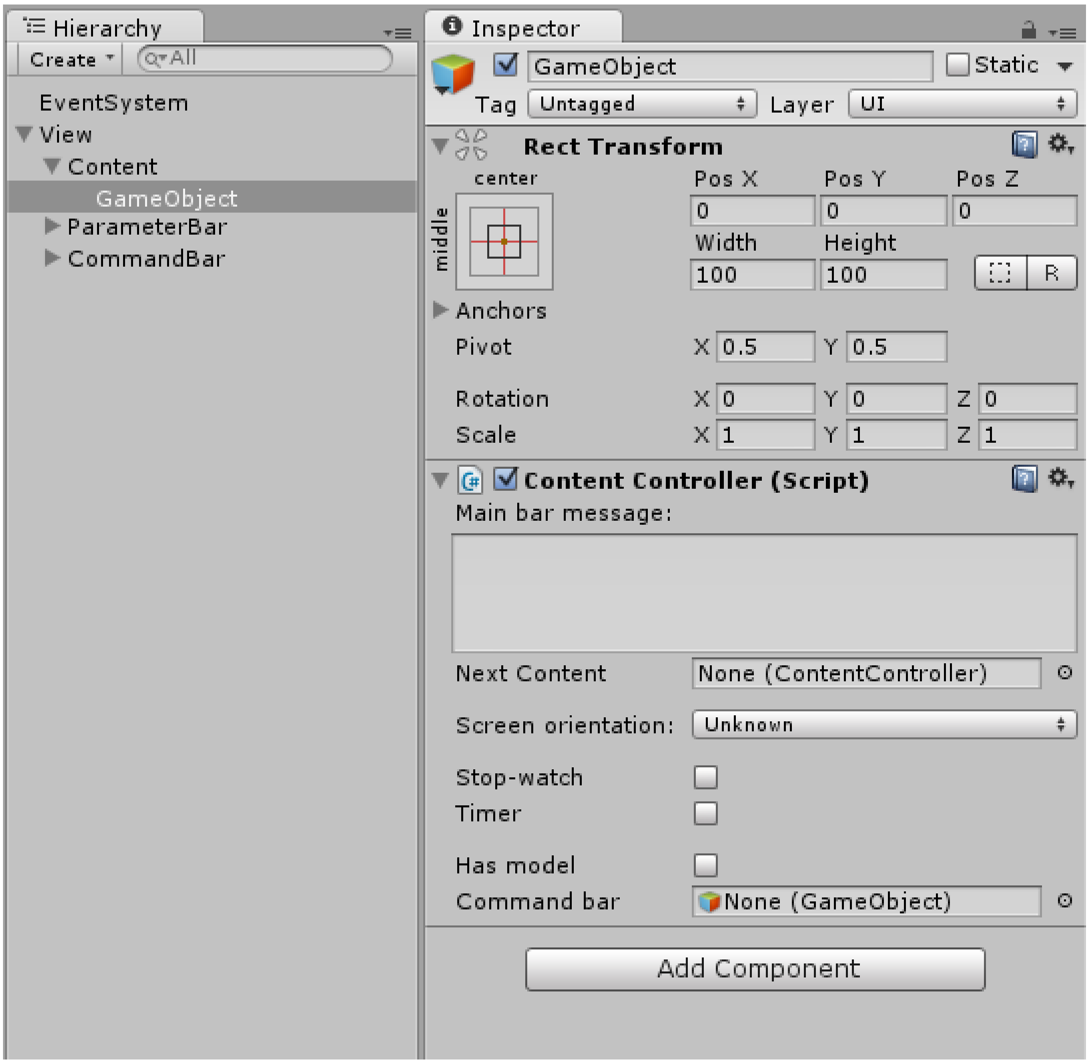
To determine which content should be displayed first, you should drag and drop the desired content to the field 'starting content' of the 'Content' component in the Hierarchy window.

3.3.1.1 ContentController

After attaching the ContentController to your content, you will notice that a new set of parameters is displayed in the Inspector, when you select your content. See picture 3.2 for demonstration.

The Rect Transform component serves as a positioner and layout organiser. The default settings, as displayed in picture 3.2, will create a rectangle a 100 points large. Usually, it is desired to fit the content to the size of the screen. You can do this by clicking the red-cross near the top-left corner of the component ui, and selecting the item in the bottom right corner of the displayed table. The, you should set all the margins to 0, so that the content is fitted to the sides of the screen. See picture 3.3 for reference of how the result should look like.

Now for the ContentController settings. As you can see in picture 3.2, the first parameter is a large text-area. In this area, you can type a message that will be displayed in the main



Hierarchy

Create ▾ All

- EventSystem
- ▼ View
  - ▼ Content
    - GameObject
  - ▶ ParameterBar
  - ▶ CommandBar

Inspector

☑ GameObject □ Static

Tag: Untagged Layer: UI

▼ Rect Transform

center / middle

Pos X	Pos Y	Pos Z
0	0	0
Width	Height	
100	100	

▶ Anchors

Pivot X 0.5 Y 0.5

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

▼ ☑ Content Controller (Script)

Main bar message:

Next Content: None (ContentController)

Screen orientation: Unknown

Stop-watch □

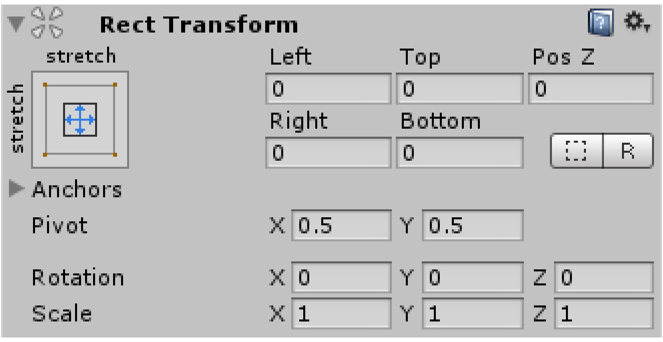
Timer □

Has model □

Command bar: None (GameObject)

Add Component

Figure 3.2: The inspector window of a newly created content, after the ContentCntroller has been attached



▼ Rect Transform

stretch / stretch

Left	Top	Pos Z
0	0	0
Right	Bottom	
0	0	

▶ Anchors

Pivot X 0.5 Y 0.5

Rotation X 0 Y 0 Z 0

Scale X 1 Y 1 Z 1

Figure 3.3: The Rect Transform component set to fit the content to the size of the screen.

bar of the application. The message will be displayed in the place of the dummy text 'MainBarMessage', which you can see in the Scene view, in the area between the large blue buttons labeled 'Back' and 'Next'. This is the main bar. You can use Unity rich-text tags here. These are basic tags for text formating. The supported tags are:

- <b>bold text</b>
- <i>italic text</i>,
- <color=blue> color set to blue</color>.
- <size=24>size set to 24</size>

By changing the text 'blue' or '24', you can receive appropriately different results. Use the <size> tag with caution though, as it can overflow from the area of the main bar, especially if the application is deployed on a mobile device with a small screen.

The next parameter is labelled 'next content'. This field specifies which content should be displayed, after clicking the 'next' button in the main bar, while the currently selected content in the Hierarchy is displayed. If you leave this field empty, the application will automatically disable the 'next' button. The 'back' button is automatically showing a previously displayed content, and thus doesn't have a setting field.

Attaching the next content happens in the same way, as attaching the 'first content' of the View component. Simply drag and drop the content from the hierarchy menu to the 'Next Content' field. After doing so, an optional 'Next content state' parameter will appear. This parameter serves as a convenient method of passing arguments from one content to another, and is useful when creating complex custom content.

Screen orientation is used when the application is deployed on a mobile device. This parameter specifies if, when the content is displayed, should automatically rotate the screen to a certain position. By leaving the default 'unknown' setting, the screen will rotate automatically after detecting the position of the device.

Stop watch and Timer can be displayed in the content. They will be displayed in the menu bar instead of the 00:00:000 and 99:99:999 texts, respectively. Timer has additional settings, after you enable it. You can specify the starting time of the timer in minutes and seconds, and also an automatic redirect after the timer expires. This is again done in the same fashion as assigning the 'Next Content' by dragging and dropping the appropriate content to the parameter field. You can also again specify the target state.

The has model parameter parameter defines that you will be adding a Modelica model to this content, and triggers certain routines when the content is displayed. You can also specify the content name after enabling this parameter. This functionality is further discussed in section 3.5.

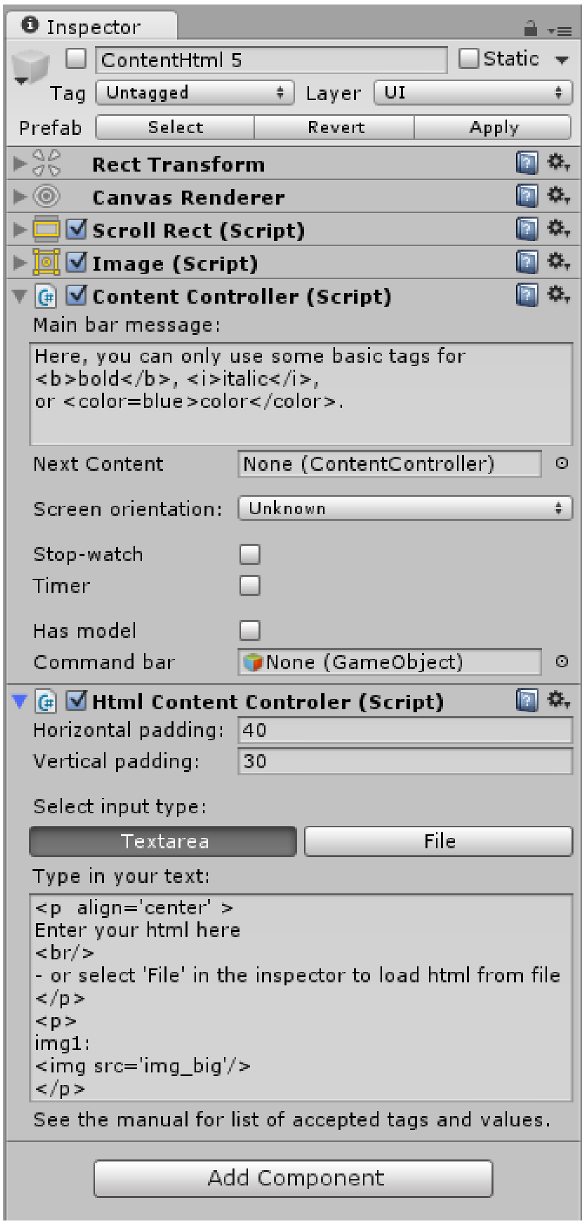
Command bar accepts objects from the Assets/Resources/ContentPrefab folder, that start with 'Actions...'. This specifies what should be displayed in the extended command bar above the main bar. The assignment is done in the same way as the 'Next content' parameter, only now you are not dragging and dropping and object from the Hierarchy window, but from the Project window, because you are not assigning an object from the application UI, but a reference to a pre-made structure. This is further discussed in the section 3.4.

3.3.2 Html content

Instead of creating a new content from scratch, a convenient template for displaying html documents is available in the Assets/Resources/ContentPrefab folder in the Project window. The object named ’ContentHtml’ is a pre-made structure that allows simple adding of new content that can display text and pictures.

By dragging this object to the ’Content’ object in the Hierarchy window, a copy of it is created, that you can edit in the same fashion as if you created a new content yourself. In the Inspector window, several more elements are present, which you can ignore, and fold by pressing the small arrow in the left of the header of the component. See picture 3.4.

The Content Controller component is present again, which serves the same purpose as in any other content.



□ ContentHtml 5 □ Static

Tag Untagged Layer UI

Main bar message:

```
Here, you can only use some basic tags for
<b>bold</b>, <i>italic</i>,
or <color=blue>color</color>.
```

Next Content None (ContentController)

Screen orientation: Unknown

Stop-watch □

Timer □

Has model □

Command bar None (GameObject)

Horizontal padding: 40

Vertical padding: 30

Select input type:

Type in your text:

```
<p  align='center' >
Enter your html here
<br/>
- or select 'File' in the inspector to load html from file
</p>
<p>
img1:
<img src='img_big'/>
</p>
```

See the manual for list of accepted tags and values.

Figure 3.4: The inspector window of a content created from the ContentHtml prefab in the Assets/Resources/ContentPrefab project folder.

3.3.2.1 Html Content Controler

Furthermore, a Html Content Controler is present below the Content Controller. Here, you can edit the settings of the displayed html content. The horizontal and vertical padding parameters specify a margin from the edges of the screen, so that the text has a frame around itself, which makes it easier to read.

The main part of the Html Content Controler is the Input. Firstly, you have to select whether you want to write the text directly, or you want to reference a file.

The RadApp framework only supports a subset of html tags to allow for basic text formatting. Currently supported tags with basic syntax examples (parameters in brackets are optional):

Link - <a href=’link’>link text</a>

Image - <img src=’imagesource’ [width=”] [height=”] />

Paragraph - <p [align=”][valign=”]> </p>

Line break - <br/>

Font format - <font [face=”] [size=”] [color=”]>

Code format - <code></code> same as <font face=’code’>

Bold - <b></b>

Italic - <i></i>

Underline - <u></u>

Strike - <s></s>

Text effect - <effect></effect>

Division - <div></div> - this element does not always render correctly.

It is highly recommended not to use any tags that are not listed, as they can break the structure of the document completely. This list applies to both text input, or the referenced file.

Included files must be copied into the Resources folder in the RadApp project Assets. You should type the whole relative path with the resources folder as a root. Do not include the file extension. (e.g. /CustomContent/MyContent/file).

The supported file types are:

- txt
- html
- htm
- xml
- bytes
- json
- csv
- yaml
- fnt

We recommend using either txt, html or htm file type to write your html text into.

3.3.2.2 Text linking

The framework also provides interface for in-text links. To handle the link redirection, it is necessary to create a custom controller. This requires at least basic programming skills and knowledge of the C# language. You can study an example implementation of this feature in the Assets/Resources/ContentPrefab/Examples/ContentHtml_ExampleLinks prefab. After dragging this object into the Content in the Hierarchy window, you will create 2 new contents, which are located in a common component 'HtmlContentLinksExample'.

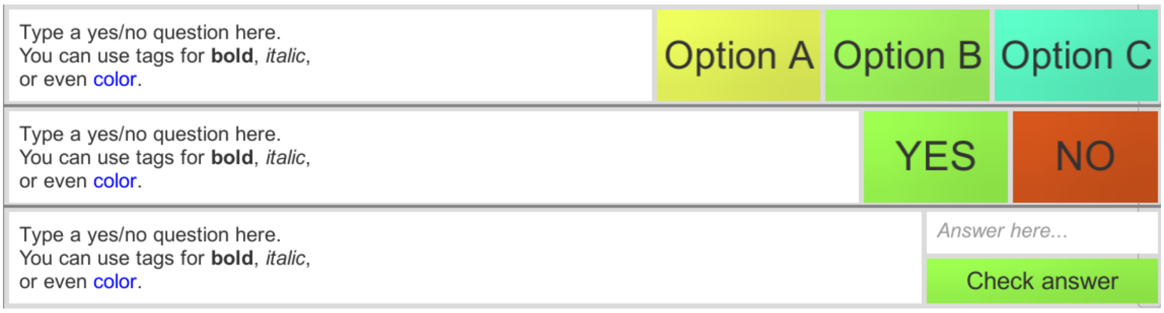
3.4 Coomand bar

The command bar is an extension of the main bar, which is normally hidden. If you assign any object from the Resources/ContentPrefabs, starting with 'Actions...', to the field Command bar in the Content Controler, you will see a new set of settings appear. These settings specify how the content of the command bar will look like.

Currently available types of the command bar content are:

- ActionsQuizABS - an A/B/C quiz question
- ActionsQuizYN - Yes/No quiz question
- ActionsTextAnswer - Text-input answer

You can see how each type will look like in the finished application in picture 3.5.



Type a yes/no question here.
You can use tags for **bold**, *italic*,
or even color.

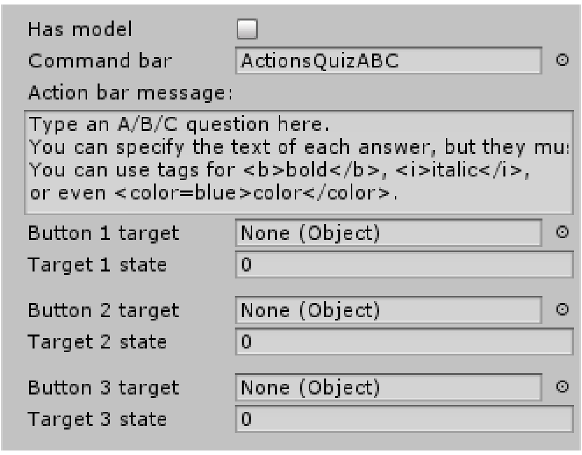
Type a yes/no question here.
You can use tags for **bold**, *italic*,
or even color.

Type a yes/no question here.
You can use tags for **bold**, *italic*,
or even color.

Figure 3.5: Prepared action bar content. From the top: an A/B/C question, a Yes/No question and a Text-input question, with the default parameters set.

The new settings that appear after assigning a selected action type, start with an 'Action bar message'. If you compare the default text in the picture 3.6 with the final look of the action bar in the picture 3.5, you can see where the message is displayed. You can also specify targets (and target states) of the actions, for each button. For the A/B/C type, you can also change labels of the buttons. For the 'Text answer' type, you can also specify what answers will be accepted as correct - after specifying the number of possible answers.

You can assign any content to each action button. If you wish to remain in the same content (stay at the same place), simply leave the button target parameter for the particular button empty. Since you can assign ANY content, you can create arbitrary content structure in your application. The 'quiz' actions do not necessarily need to contain a 'test' questions, but can also serve simply to decide which content should be displayed next. Therefore, you can leave the 'next content' parameter empty.



```
Has model          □
Command bar        ActionsQuizABC
Action bar message:
Type an A/B/C question here.
You can specify the text of each answer, but they mu:
You can use tags for <b>bold</b>, <i>italic</i>,
or even <color=blue>color</color>.
Button 1 target    None (Object)
Target 1 state     0

Button 2 target    None (Object)
Target 2 state     0

Button 3 target    None (Object)
Target 3 state     0
```

Figure 3.6: The content controller settings bottom part, after assigning a command bar action content.

3.5 Modelica model

Creating and connecting a model to the application requires a deep understanding of programming and mathematical modelling principles. It is not the purpose of this guide to teach these principles, and the RadApp framework merely provides necessary interfaces and describes the tasks to reduce, or possibly remove, any necessary research.

This part is not intended for beginner level application creators, and thus only the RadApp specific tasks will be explained.

3.5.1 Creating model

Creating the model is possible in any Modelica editor, but for the purposes of the Bodylight and the RadApp framework, it is highly necessary to have Open Modelica 1.9.1. Even the specific version is given for purpose. Saving the model in Open Modelica 1.9.2 already creates incompatibilities later in the RadApp connection process.

3.5.1.1 Translating model

In the RadApp root, you will find a 'ModelTranslator' folder. Here, you will find an example modelica file, a batch file, and a mos file. The model is purely demonstrational, and has no actual content. There is also an F# class file, which we will use later.

The mos file provides a necessary preprocessing, and the batch file calls OpenModelica C# translator. In the mos file, you should edit the paths to your model file, as well as in the batch file, where you also have to specify your Open Modelica binary location (or use PATH variables). This approach is intended for and tested on Windows OS.

3.5.2 Bodylight interface

After translating the model, you can create a new Solution in Visual Studio, in which you should include Bodylight.Simulation, located in Bodylight/Simulation, and Bodylight.ViewModels and UnityViewService projects located in Bodylight/Presentation folder (Bodylight being the root folder of Bodylight framework). Create a new solution build configuration named 'Unity Debug', and set both included projects to the same value, which should be provided in their configuration list.

Now you have to create two new projects.

Firstly, a C# class library project that will contain you translated C# model. Set up the target framework to .NET 3.5. In Bodylight framework it is common to call these projects by the pattern 'ExampleUnityModels'. Then add your translated model into this project, and add both Bodylight simulation and ViewModels projects as references.

Secondly, an F# library project, again targeting .NET 3.5, where you will have to create a custom interface that you will use in Unity. The naming convention for this project is 'ExampleUnityViewModels', and a singular form applies to the F# file in which you will create the interface. The interface has to inherit a NotifyBase() type, be in a Bodylight.Presentation.ExampleUnity namespace, and standard includes are System, Bodylight.Common, Bodylight.Solvers and Bodylight.ViewModels.

Implementation and detailed usage of the Bodylight framework is not part of this guide. Therefore, only a simple example is provided in RadApp/ModelTranslator/ExampleUnityViewModel.fs file.

After implementing the ViewModel interface, you can build the solution.

3.5.3 Adding model to Unity

Now that you have finished preparing the Bodylight part, you should add necessary dll files to the Unity project. Place them anywhere in the Assets/Resources/CustomContent/. It is convenient to crate a new folder for your particular content, and a subfolder named 'Libraries'.

You will have to copy Bodylight.Simulation.dll, Bodylight.ViewModels.dll, ExampleUnityModels.dll, ExampleUnityViewModels.dll and also UnityViewService.dll. You do not have to add the FSharp.Code.dll file, as it is already present in the Codebase/Libraries.

If you are including multiple models, only add Bodylight.Simulation, Bodylight.ViewModels and UnityViewService once. Unity will automatically add references to any included dll libraries to the Mono project.

Now you have to implement the ModelVariableProvider abstract class. In its 'Intialise' method, call the static method `UnityTickProvider.ProvideTick (this)`, and create a new instance of your ExampleUnityViewModel.

Furher in the Initialise method you should create a new Dictionary<string, ModelVariable> and assign in into the 'base.variables' variable. Then you can create appropriate ModelVariable instances, that will mimic the necessary variables in your model, and add them to the dictionary. Study the documentation of each ModelVariable implementation to discover which type matches your model variables the best. If your model has any input values, you can create methods called when the ModelVariable value changes. This can be done using a ModelVariable.Get...SlotControler.Get...ValueChangedEvent (replace the dotted parts accordingly to the ModelVariable type you are using) method. The returned event is invoked each time user manipulates the visual components of the ModelVariables (if they have any).

Create an 'Update' method with no parameters and void return value. This method is called by Unity before rendering each frame. Here, you should firstly call the static UnityTickProvider.ProvideTick method. Then, you can update each output ModelVariable with the values from the ViewModel. Be sure not no overwrite the input values here.

Assign your ModelVariableProvider implementation to your content. Make sure to assign it to the same GameObject that holds the ContentController component, and check the 'Has model' property there.

Now, RadApp will automatically create visual controls for the ModelVariables that are present dictionary, and shows them in the side bars.

3.5.4 Creating custom visualisation

Adding the model is the first step, but you should also create a visualisation to present how the model works. To access the model variables, create a new csharp class (call it ExampleContentControler) that will inherit MonoBehaviour base class, which will allow you to add as a component to your content GameObject. In this class, you can reference your ExampleViewModel by creating a public variable, and then assigning the ExampleModelVariableProvider to it in the Unity Inspector.

Now you can create a custom visual content and specify how it should react to changes in the model in your ExampleContentControler class. This obviously requires at least basic understanding of Unity. For better work, you can disable the parameter bars and the command bar, but you have to re-enable them before entering the game mode, or building the application. Also be aware that the side bars will block parts of the screen when the application runs.

3.6 Building and deploying

To build the application for a specific platform, select 'File' -> 'Build Settings' in the Unity top menu.

You will see a new window, in which you can select the target platform and edit some basic settings. To start the build, press the 'Build' button and select an output directory, after which you will see a progress-bar of the build process. When the build process finishes, the target folder will open.

3.6.1 Pre-build tasks

Before building the application, it is recommended to go through several tasks to ensure that the build will be successful, and the application will work.

Firstly, try running the application in the Unity Game mode first and going through all the contents of your application. This simple test will reveal any bugs or errors in the design. If there is any major problem, that prevents the application from working, you will see a highlighted output in the Console window.

To reduce the final package size, you can remove all unused files, like pictures, from the CustomContent folder. If you have no model attached, you can also remove the FSharp.Core file from Codebase/Libraries. Do not remove any files from the rest of the Codebase or Editor folder, or from Resources/ContentPrefabs folder unless you are skilled programmer and familiar with Unity. You could break the framework structure irreversibly!

3.6.2 Build options

For each platform there are specific build options, which will show in the Inspector after pressing the 'Player settings' button in the 'Build settings' window.

Common options are company and product name, which you can change freely, as well as the application default icon, cursor, or cursor hotspot.

You can select settings specific for each platform by clicking appropriate picture in the Inspector panel.You can change the icons, default resolution, bundle identifier or version, but if you do not understand the meaning of the setting, you should leave it at the default value.

You can find detailed description of any Unity feature in the Unity documentation, available online at docs.unity3d.com.

3.6.3 Building for Windows

There are two possibilities of build for windows. You can either build a standalone executable, or a Windows 8/8.1 Store/Phone app. If you are not familiar with the Windows Store/Phone applications, and only wish to create a simple program that you can run directly, choose the standalone option.

3.6.3.1 Standalone

Select 'PC and Mac standalone' from the Platform list in the Build Settings window. Select 'Widnows' as the target platform. For more compatibility, select x86 architecture. You can leave all the checkboxes un-checked and start the build process.

After the building is complete, you will find an `application_name`.exe file with a RadApp (or your custom) icon, and an `application_name_Data` folder in the build target folder. Do not separate the exe file from the _ Data folder, or the application will not work.

You can run your application by clicking the exe file.

3.6.3.2 Windows Phone and Store

Building for Phone/Store requires an extra step in the Visual Studio. After finishing the build process in unity, open the generated project in the build target folder in Visual Studio. Right click the project in the Solution Explorer and from the drop-down menu select 'Store' -> 'Create App Packages' and follow instruction in the newly opened window.

3.6.4 Building for Android

Building for android creates a single apk file, which can be then directly used for distribution or tested in an Android device, by transfering it into the device (e.g. via usb cable) and installing it directly from the file.

Chapter 4

Appendix

This guide admittedly does not explain details of basic Unity principles, or any other mentioned software. Unity provides excellent beginner level tutorials, and has an extensive online manual. The same goes for the other tools. It is expected that advanced applications are created by advanced users, and thus some features are described only briefly, and some, on the other hand, maybe even in too much detail.

Hopefully, the guide covered the basic use well enough to explain main use of the RadApp framework.

Appendix C

RadApp technical reference

RadApp

0.3.1

Technical Reference

Generated by Doxygen 1.8.9.1

28. April 2015

Contents

Chapter 1

Namespace Index

1.0.1 Namespace List

Here is a list of all documented namespaces with brief descriptions:

1.1 Hierarchical Index

1.1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.0.2 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Namespace Documentation

3.0.3 Package RadApp

Namespaces

- package **Controllers**
- package **ModelInterface**
- package **Utils**

3.0.4 Package RadApp.Controllers

Namespaces

- package **Private**
- package **Public**

3.0.5 Package RadApp.Controllers.Private

Namespaces

- package **Editors**
- package **Plugins**

Classes

- class **ActionsPrefabControler**

  Contorler attached to any prefab that serves as possible content of the actionBar. In servers as a provider of the prefab parameters to the custon inspector of ContentControler. When the prefab is attached to some content, the custom inspector automatically displays fields for customisation of the future instance of this prefab. There are also methods for the instantiated version, which serve to set the properties of the object when it is being displayed as the actionBar content.
- class **ActionsTextInputPrefabControler**

  Extension of the ActionsPrefabController that is used for command bar content that accepts textual answers.

- class **CommandBarControler**

  *Class managing the command bar itself. Through this class the command bar can reach the **SceneDisplayControler** (*p. *30) to send signals when any command button is pressed. If there is any content in the command bar, and it has buttons, these buttons are comunicationg with the SceneDisplayControelr directly, not through this class.*
- class **MainMenuControler**

  Main menu controler, probably only used in the StartScene
- class **ParameterBarControler**

  Class that manages the parameter bars, fills them with given variable visual components,
- class **SceneDisplayControler**

  This class controls which content will be shown in the scene. It has access to the current content controler, holds stack of previously displayed contents and also does all the routines related to displaying new content. That includes toggle of the newx/prev buttons, action bar setting, main bar message and timers, and the side bar settings.
- class **StaticGraphControler**

  This class is a wrapper for the StaticGraph utility, that creates a more convenient interface and provides setters of value labels and axes names.

3.0.6 Package RadApp.Controllers.Private.Editors

3.0.7 Package RadApp.Controllers.Private.Plugins

3.0.8 Package RadApp.Controllers.Public

Classes

- class **ContentController**

  Class controling the currently displayed content. It has pointer to the next content. This script has its own inspector editor attached in Editor/ContentControlerEditor, which provides the main content settings interface in the Unity editor. ContenControlerEditor

3.0.9 Package RadApp.ModelInterface

Namespaces

- package **ModelVariables**
- package **SlotControlers**

Classes

- class **ModelVariableProvider**

  Model variable provider abstract class. Implement this to provide all the model variables instances necesarry in your model. Here is also where any remaining logic should be stored, because all the other classes are common for all contents. Provided methods can be overridden, if yoy wish to use other structure than dictionary for organising your model variable instances.

3.0.10 Package RadApp.ModelInterface.ModelVariables

Classes

- class **ModelBooleanVariable**

  Output implementation for interactive interval variable with slider.
- class **ModelGraphOutputVariable**

  Output implementation
- class **ModelIntervalVariable**

  Output implementation for interactive interval variable with slider.
- class **ModelOutputVariable**

  The simplest Modelvariable implementation. This type holds a float value without any graphical component. Use this variable, if you dont want its value to be displayed in the parameter bars, but you need to use it in your graphical content. If you need another type (bool, object), you can use one of the Interactive implementations, and disable the graphical component in the specific instance.
- class **ModelTextOutputVariable**

  Output implementation

3.0.11 Package RadApp.ModelInterface.SlotControlers

Classes

- class **SlotInteractiveControler**

  A placeholder class in case it is necesarry to manipulate all interactive controlers as one type, or extract commont functionality.

3.0.12 Package RadApp.Utils

Namespaces

- package **Events**
- package **Exceptions**
- package **Graphs**
- package **Html**

Classes

- class **StopWatch**

  Record time until Stop function is called. Recording can then be returned or reseted with Start or Reset functions respectfuly. You have to call the StartRecording function after initialisation manually to start recording time.
- class **Timer**

  ***Timer** (*p. *36) class that counts down from preseted time in minues or seconds. Use the SetTime function to initialise the **Timer** (*p. *36). You have to manually call StartCounting funtion to start the timer.*

3.0.13 Package RadApp.Utils.Events

Classes

- class **LinkClickedEvent**

  Event used in the HTML render, that is fired when a link is clicked.
- class **ScreenOrientationChangedEvent**

  Event used in ScreenOrientationDetector, that is fired when a screen orientation change is detected.
- class **SelectChangedEvent**

  Event used in the SlotSelectControler that is fired when the selection has changed.
- class **SwipeDetectedEvent**

  Event used in SwipeDetector, that is fired when a swipe across the screen is detected

3.0.14 Package RadApp.Utils.Exceptions

Classes

- class **VariableNotFoundException**

  Variable not found exception is used ModelVariableProvider to be invoked when there has been a request for a variable, whose name does not exist in the local dictionary.
- class **VariableProviderNotAttachedException**

  Variable provider not attached exception, used during the content manipulation
- class **WrongVariableTypeException**

  Wrong variable type exception is used in ModelVariable and SlotControlers to be invoked when a wrong value type or event has been accessed.

3.0.15 Package RadApp.Utils.Graphs

Classes

- class **StaticGraph**

  Time graph is class that creates visual time-dependent graph of given variables. Use DrawValue to draw the next value. Time resolution is not currently controlled by time, but by the frequency with which the DrawValue function os called (usually in an Update function of another script). If you want specific refresh rate, use timing function to controll the frequency externally (e.g. using MonoBehaviour.InvokeRepeating).
- class **TimeGraph**

  Time graph is class that creates visual time-dependent graph of given variables. Use DrawValue to draw the next value. Time resolution is not currently controlled by time, but by the frequency with which the DrawValue function os called (usually in an Update function of another script). If you want specific refresh rate, use timing function to controll the frequency externally (e.g. using MonoBehaviour.InvokeRepeating).

3.0.16 Package RadApp.Utils.Html

Classes

- class **HtmlDevice**

  Provides gate between HTMLEngine and Unity3D. Implements abstract class.
- class **HtmlFont**

  Provides font for use with HTMLEngine. Implements abstract class.
- class **HtmlImage**

  Provides image for use with HTMLEngine. Implements abstract class.

Chapter 4

Class Documentation

4.0.17 RadApp.Controllers.Private.ActionsPrefabControler Class Reference

Contorler attached to any prefab that serves as possible content of the actionBar. In servers as a provider of the prefab parameters to the custon inspector of ContentControler. When the prefab is attached to some content, the custom inspector automatically displays fields for customisation of the future instance of this prefab. There are also methods for the instantiated version, which serve to set the properties of the object when it is being displayed as the actionBar content.

Inherits MonoBehaviour.

Inherited by **RadApp.Controllers.Private.ActionsTextInputPrefabControler**.

Public Member Functions

- int **CountButtons** ()

  Counts the buttons.
- string **GetText** ()

  Gets the main text.
- bool **HasCustomLabels** ()

  Checks whether custom labels for containing buttons are allowed.
- void **SetText** (string text)

  Sets the main text. USE THIS METHOD ONLY ON INSTANTIATED OBJECTS - othwrwise it will change the prefab default settings.
- void **SetButtonText** (int index, string text)

  Sets the button text (i.e. label).
- virtual void **SetButtonTarget** (int index, GameObject target, int state)

  Sets the button target.

4.0.17.1 Detailed Description

4.0.17.2 Member Function Documentation

4.0.17.2.1 int RadApp.Controllers.Private.ActionsPrefabControler.CountButtons ( )

Returns

The number of buttons in this actionBar content.

4.0.17.2.2 string RadApp.Controllers.Private.ActionsPrefabControler.GetText ( )

Returns

The main text.

4.0.17.2.3 bool RadApp.Controllers.Private.ActionsPrefabControler.HasCustomLabels ( )

Returns

`true`, if custom labels are allowed, `false` otherwise.

4.0.17.2.4 virtual void RadApp.Controllers.Private.ActionsPrefabControler.↩ SetButtonTarget (int index, GameObject target, int state) [virtual]

Parameters

index	Index.
target	Target.

Reimplemented in **RadApp.Controllers.Private.ActionsTextInputPrefabControler** (p. 15).

4.0.17.2.5 void RadApp.Controllers.Private.ActionsPrefabControler.SetButtonText (int index, string text)

Parameters

index	Index of button you wish to change the label of.
text	New text of the button.

4.0.17.2.6 void RadApp.Controllers.Private.ActionsPrefabControler.SetText (string text)

Use these methods in the INSTANTIATED objects only, otherwise it will change the prefab default settings

Parameters

text	Text to be displayed as the main message.

4.0.18 RadApp.Controllers.Private.ActionsTextInputPrefabControler Class Reference

Extension of the ActionsPrefabController that is used for command bar content that accepts textual answers.

Inherits **RadApp.Controllers.Private.ActionsPrefabControler**.

Public Member Functions

- override void **SetButtonTarget** (int index, GameObject target, int state)

  Sets the button target.

4.0.18.1 Detailed Description

4.0.18.2 Member Function Documentation

4.0.18.2.1 override void RadApp.Controllers.Private.ActionsTextInputPrefab↩ Controler.SetButtonTarget (int index, GameObject target, int state) [virtual]

Parameters

index	Index.
target	Target.

Reimplemented from **RadApp.Controllers.Private.ActionsPrefabControler** (p. 14).

4.0.19 RadApp.Controllers.Private.CommandBarControler Class Reference

Class managing the command bar itself. Through this class the command bar can reach the **SceneDisplayControler** (p. 30) to send signals when any command button is pressed. If there is any content in the command bar, and it has buttons, these buttons are comunicationg with the SceneDisplayControelr directly, not through this class.

Inherits MonoBehaviour.

Public Member Functions

- void **SetAndStartTimer** (int minutes, int seconds)

  Sets, shows and starts the timer at given time.
- void **SetAndStartStopWatch** ()

  Sets, shows and starts the stopwatch at time 0.
- void **DisableTimer** ()

  Hides the timer.
- void **DisableStopWatch** ()

  Hides the stopwatch
- void **SetMainMessage** (string text)

  Sets the main message on the mainBar.
- void **SetActions** (GameObject actionsPrefab, string message, GameObject[] button↩ Targets, int[] targetStates, string[] buttonLabels, string[] answers)

  this method is used to set the content of the actionBar. If there already is any content present, it will be discarted first.
- void **ResetActions** ()

  Removes the currently displayed actionBar content.

- void **SetTimers** (bool timer, int mins, int secs, bool stopWatch, GameObject target, int targetState)

  Sets the timer and stopwatch to given values.
- void **ExitButtonPressed** ()

  Returns to the main menu (StartScene).
- void **HideButtonPressed** ()

  Hides the actionBar, if it is visible, or shows it, if it is hidden. Also toggles the default button to act accordingly.
- void **NextButtonPressed** ()

  Shows next content (if there is any).
- void **PrevButtonPressed** ()

  Shows previous content (if there is any).
- void **DisableNextButton** ()

  Disables the 'next' button.
- void **DisablePrevButton** ()

  Disables the 'prev' button.
- void **EnableNextButton** ()

  Enables the 'next' button.
- void **EnablePrevButton** ()

  Enables the 'prev' button.

4.0.19.1 Detailed Description

are pressed.

4.0.19.2 Member Function Documentation

4.0.19.2.1 void RadApp.Controllers.Private.CommandBarControler.DisableNextButton ( )

4.0.19.2.2 void RadApp.Controllers.Private.CommandBarControler.DisablePrevButton ( )

4.0.19.2.3 void RadApp.Controllers.Private.CommandBarControler.DisableStopWatch ( )

4.0.19.2.4 void RadApp.Controllers.Private.CommandBarControler.DisableTimer ( )

4.0.19.2.5 void RadApp.Controllers.Private.CommandBarControler.EnableNextButton ( )

4.0.19.2.6 void RadApp.Controllers.Private.CommandBarControler.EnablePrevButton ( )

4.0.19.2.7 void RadApp.Controllers.Private.CommandBarControler.ExitButtonPressed ( )

4.0.19.2.8 void RadApp.Controllers.Private.CommandBarControler.HideButtonPressed ( )

4.0.19.2.9 void RadApp.Controllers.Private.CommandBarControler.NextButtonPressed ( )

4.0.19.2.10 void RadApp.Controllers.Private.CommandBarControler.PrevButtonPressed ( )

4.0.19.2.11 void RadApp.Controllers.Private.CommandBarControler.ResetActions ( )

4.0.19.2.12 void RadApp.Controllers.Private.CommandBarControler.SetActions (GameObject actionsPrefab, string message, GameObject[] buttonTargets, int[] targetStates, string[] buttonLabels, string[] answers)

Parameters

actionsPrefab	The gameObject used as the command bar content
message	The text to be dispalyed as the command bar content as the main message
button↩ *Targets*	Array of target contents of the buttons in the command bar content.
targetStates	Target states of the command bar buttons
buttonLabels	Labels of the command bar buttons
answers	Accepted answers. Only has effect with the text answer content

4.0.19.2.13 void RadApp.Controllers.Private.CommandBarControler.SetAndStartStop↩ Watch ( )

4.0.19.2.14 void RadApp.Controllers.Private.CommandBarControler.SetAndStartTimer (int minutes, int seconds)

Parameters

minutes	Minutes.
seconds	Seconds.

4.0.19.2.15 void RadApp.Controllers.Private.CommandBarControler.SetMainMessage (string text)

Parameters

text	The text to be displayed.

4.0.19.2.16 void RadApp.Controllers.Private.CommandBarControler.SetTimers (bool timer, int mins, int secs, bool stopWatch, GameObject target, int targetState)

Parameters

timer	If set to `true` timer will be displayed
mins	Minutes to start the timer at
secs	Seconds to start the timer at
stopWatch	If set to `true` stop watch will be displayed
target	Target content to redirect to when the timer expires
targetState	State of the content displayed when timer expires

4.0.20 RadApp.Controllers.Public.ContentController Class Reference

Class controling the currently displayed content. It has pointer to the next content. This script has its own inspector editor attached in Editor/ContentControlerEditor, which provides the main content settings interface in the Unity editor. ContenControlerEditor

Inherits MonoBehaviour.

Public Member Functions

- GameObject **GetActionsPrefab** ()

  Actions prefab if the prefab which should be instantiated and displayed in the actionBar when displaying the content that this script is attached to.
- string **GetActionsMessage** ()

  *The message that should be displayed as the main text of the actionBar content instantiated from **GetActionsPrefab()** (p. 20);*
- string[] **GetActionsButtonLabels** ()

  If the attached actionBar content has any buttons, this method gives you an array of strings to be displayed respectively as the button labels.
- GameObject[] **GetActionsButtonTargets** ()

  If the attached actionBar content has any buttons, this method gives you an array of GameObjects that should be set respectively as the button onClick targets.
- int[] **GetActionsButtonTargetStates** ()

  If the attached actionBar content has any buttons, this method gives you an array of ints that should be set respectively as the button onClick targets state.
- string[] **GetActionsAnswers** ()

  Gets the accepted answers, if the used action prefab has them specicfied.
- void **SetState** (int state)

  Sets the state. The state property of this controlelr can be used to manupulate the content directly by checking the property in the Update method.
- int **GetState** ()

  Gets the currently set state.
- bool **HasModel** ()

  Checks, whether this content has any model attached.
- string **GetModelName** ()

  Gets the name of the model.
- void **ShowContent** ()

Shows the content that this script is attached to.

- void **HideContent** ()

  Hides the content that this script is attached to.

- bool **HasNextContent** ()

  Determines whether the content that this script is attached to has next content proprety set. It doesnt check whether the next content property has required parameters (such as this script attached to it).

- **ContentController GetNextContentControler** ()

  Gets the ContentControler component of the next content.

- int **GetNextContentState** ()

  Gets the desired state of the next content.

- string **GetContentName** ()

  Gets the name of the content.

- **ModelVariableProvider GetModelVariableProvider** ()

  Gives the model variable provider, which holds all the model variables.

- ScreenOrientation **GetScreenOrientation** ()

  Gets the screen orientation that user specified as the prefference for this content.

- bool **HasTimer** ()

  Determines whether this content should display timer.

- bool **HasStopWatch** ()

  Determines whether this content should dispaly stop watch.

- int **GetTimerMinutes** ()

  Gets the remaining minutes on the timer.

- int **GetTimerSeconds** ()

  Gets the remaining seconds on the timer.

- GameObject **GetTimerTarget** ()

  User can specify a content to redirect to when the timer expires. This methdod returns that content.

- int **GetTimerTargetState** ()

  If there is specified target for timer expiration, user can also optionally specify the target state, that will be passed as a parameter to the target content.

4.0.20.1 Detailed Description

4.0.20.2 Member Function Documentation

4.0.20.2.1 string [] RadApp.Controllers.Public.ContentController.GetActionsAnswers ( )

Returns

The actions accepted answers.

4.0.20.2.2 string [] RadApp.Controllers.Public.ContentController.GetActionsButtonLabels ( )

Returns

The actions button labels.

4.0.20.2.3 GameObject [] RadApp.Controllers.Public.ContentController.GetActions↩ButtonTargets ( )

Returns

The actions button targets.

4.0.20.2.4 int [] RadApp.Controllers.Public.ContentController.GetActionsButtonTarget↩States ( )

Returns

The actions button target states.

4.0.20.2.5 string RadApp.Controllers.Public.ContentController.GetActionsMessage ( )

Returns

The actions message.

4.0.20.2.6 GameObject RadApp.Controllers.Public.ContentController.GetActionsPrefab ( )

Returns

The actions prefab.

4.0.20.2.7 string RadApp.Controllers.Public.ContentController.GetContentName ( )

Returns

The content name.

4.0.20.2.8 string RadApp.Controllers.Public.ContentController.GetModelName ( )

Returns

The model name.

4.0.20.2.9 **ModelVariableProvider** RadApp.Controllers.Public.ContentController.Get↩ModelVariableProvider ( )

Returns

The model variables provider.

4.0.20.2.10 **ContentController** RadApp.Controllers.Public.ContentController.GetNext↩ ContentControler ( )

Returns

The next content control.

4.0.20.2.11 int RadApp.Controllers.Public.ContentController.GetNextContentState ( )

Returns

The next content state.

4.0.20.2.12 ScreenOrientation RadApp.Controllers.Public.ContentController.GetScreen↩ Orientation ( )

Returns

The screen orientation.

4.0.20.2.13 int RadApp.Controllers.Public.ContentController.GetState ( )

Returns

The current state.

4.0.20.2.14 int RadApp.Controllers.Public.ContentController.GetTimerMinutes ( )

Returns

The remaining timer minutes.

4.0.20.2.15 int RadApp.Controllers.Public.ContentController.GetTimerSeconds ( )

Returns

The remaining timer seconds.

4.0.20.2.16 GameObject RadApp.Controllers.Public.ContentController.GetTimerTarget ( )

Returns

The content to be redirected to when the timer expires, or null, if there was no content specified for this action.

4.0.20.2.17 int RadApp.Controllers.Public.ContentController.GetTimerTargetState ( )

Returns

The timer target content state.

4.0.20.2.18 bool RadApp.Controllers.Public.ContentController.HasModel ( )

Returns

`true`, if content has model, `false` otherwise.

4.0.20.2.19 bool RadApp.Controllers.Public.ContentController.HasNextContent ( )

Returns

`true` if this instance has next content; otherwise, `false`.

4.0.20.2.20 bool RadApp.Controllers.Public.ContentController.HasStopWatch ( )

Returns

`true` if this content has stop watch; otherwise, `false`.

4.0.20.2.21 bool RadApp.Controllers.Public.ContentController.HasTimer ( )

Returns

`true` if this content has timer; otherwise, `false`.

4.0.20.2.22 void RadApp.Controllers.Public.ContentController.HideContent ( )

4.0.20.2.23 void RadApp.Controllers.Public.ContentController.SetState (int state)

Parameters

state	Number of the new state.

4.0.20.2.24 void RadApp.Controllers.Public.ContentController.ShowContent ( )

4.0.21 RadApp.Utils.Html.HtmlDevice Class Reference

Provides gate between HTMLEngine and Unity3D. Implements abstract class.

Inherits HtDevice.

Public Member Functions

- override HtFont **LoadFont** (string face, int size, bool bold, bool italic)

  Load font
- override HtImage **LoadImage** (string src)

  Load image
- override void **FillRect** (HtRect rect, HtColor color)

  FillRect implementation

4.0.21.1 Detailed Description

4.0.21.2 Member Function Documentation

4.0.21.2.1 override void RadApp.Utils.Html.HtmlDevice.FillRect (HtRect rect, HtColor color)

Parameters

rect	
color	

4.0.21.2.2 override HtFont RadApp.Utils.Html.HtmlDevice.LoadFont (string face, int size, bool bold, bool italic)

Parameters

face	Font name
size	Font size
bold	Bold flag
italic	Italic flag

Returns

Loaded font

4.0.21.2.3 override HtImage RadApp.Utils.Html.HtmlDevice.LoadImage (string src)

Parameters

src	src attribute from img tag

Returns

Loaded image

4.0.22 RadApp.Utils.Html.HtmlFont Class Reference

Provides font for use with HTMLEngine. Implements abstract class.

Inherits HtFont.

Public Member Functions

- **HtmlFont** (string face, int size, bool bold, bool italic)
  Ctor
- override HtSize **Measure** (string text)
  Measuring text width and height
- override void **Draw** (HtRect rect, HtColor color, string text)
  Draw method.

Public Attributes

- readonly GUIStyle **style** = new GUIStyle()
  style to draw
- readonly GUIContent **content** = new GUIContent()
  content to draw

Properties

- override int **LineSpacing** `[get]`
  Space between text lines in pixels
- override int **WhiteSize** `[get]`
  Space between words

4.0.22.1 Detailed Description

4.0.22.2 Constructor & Destructor Documentation

4.0.22.2.1 RadApp.Utils.Html.HtmlFont.HtmlFont (string face, int size, bool bold, bool italic)

Parameters

face	Font name
size	Font size
bold	Bold flag
italic	Italic flag

4.0.22.3 Member Function Documentation

4.0.22.3.1 override void RadApp.Utils.Html.HtmlFont.Draw (HtRect rect, HtColor color, string text)

Parameters

rect	Where to draw
color	Text color
text	Text

4.0.22.3.2 override HtSize RadApp.Utils.Html.HtmlFont.Measure (string text)

Parameters

text	text to measure

Returns

width and height of measured text

4.0.22.4 Member Data Documentation

4.0.22.4.1 readonly GUIContent RadApp.Utils.Html.HtmlFont.content = new GUIContent()

4.0.22.4.2 readonly GUIStyle RadApp.Utils.Html.HtmlFont.style = new GUIStyle()

4.0.22.5 Property Documentation

4.0.22.5.1 override int RadApp.Utils.Html.HtmlFont.LineSpacing `[get]`

4.0.22.5.2 override int RadApp.Utils.Html.HtmlFont.WhiteSize `[get]`

4.0.23 RadApp.Utils.Html.HtmlImage Class Reference

Provides image for use with HTMLEngine. Implements abstract class.

Inherits HtImage.

Public Member Functions

- **HtmlImage** (string source)

  Ctor
- override void **Draw** (HtRect rect, HtColor color)

  Draw method

Public Attributes

- Texture2D **Texture**

  Loaded texture

Properties

- override int **Width** `[get]`

  Returns width of image
- override int **Height** `[get]`

  Returns height of image

4.0.23.1 Detailed Description

4.0.23.2 Constructor & Destructor Documentation

4.0.23.2.1 RadApp.Utils.Html.HtmlImage.HtmlImage (string source)

Parameters

source	src attribute from img tag

4.0.23.3 Member Function Documentation

4.0.23.3.1 override void RadApp.Utils.Html.HtmlImage.Draw (HtRect rect, HtColor color)

Parameters

rect	Where to draw
color	Color to use (ignored for now)

4.0.23.4 Member Data Documentation

4.0.23.4.1 Texture2D RadApp.Utils.Html.HtmlImage.Texture

4.0.23.5 Property Documentation

4.0.23.5.1 override int RadApp.Utils.Html.HtmlImage.Height `[get]`

4.0.23.5.2 override int RadApp.Utils.Html.HtmlImage.Width `[get]`

4.0.24 RadApp.Utils.Events.LinkClickedEvent Class Reference

Event used in the HTML render, that is fired when a link is clicked.

Inherits UnityEvent< string >.

4.0.24.1 Detailed Description

4.0.25 RadApp.Controllers.Private.MainMenuControler Class Reference

Main menu controler, probably only used in the StartScene

Inherits MonoBehaviour.

Public Member Functions

- void **StartButtonPressed** ()

  Starts actions after pressing the start button.
- void **ExitButtonPressed** ()

  Starts actions after pressing the exit button.

4.0.25.1 Detailed Description

4.0.25.2 Member Function Documentation

4.0.25.2.1 void RadApp.Controllers.Private.MainMenuControler.ExitButtonPressed ( )

4.0.25.2.2 void RadApp.Controllers.Private.MainMenuControler.StartButtonPressed ( )

4.0.26 RadApp.ModelInterface.ModelVariables.ModelBooleanVariable Class Reference

Output implementation for interactive interval variable with slider.

Inherits **RadApp.ModelInterface.ModelVariables.ModelOutputVariable**.

4.0.26.1 Detailed Description

4.0.27 RadApp.ModelInterface.ModelVariables.ModelGraphOutputVariable Class Reference

Output implementation

Inherits **RadApp.ModelInterface.ModelVariables.ModelOutputVariable**.

4.0.27.1 Detailed Description

4.0.28 RadApp.ModelInterface.ModelVariables.ModelIntervalVariable Class Reference

Output implementation for interactive interval variable with slider.

Inherits **RadApp.ModelInterface.ModelVariables.ModelOutputVariable**.

4.0.28.1 Detailed Description

4.0.29 RadApp.ModelInterface.ModelVariables.ModelOutputVariable Class Reference

The simplest Modelvariable implementation. This type holds a float value without any graphical component. Use this variable, if you dont want its value to be displayed in the parameter bars, but you need to use it in your graphical content. If you need another type (bool, object), you can use one of the Interactive implementations, and disable the graphical component in the specific instance.

Inherits RadApp.ModelInterface.ModelVariables.ModelVariable.

Inherited by **RadApp.ModelInterface.ModelVariables.ModelBooleanVariable**, **RadApp.ModelInterface.ModelVariables.ModelGraphOutputVariable**, **RadApp.ModelInterface.ModelVariables.ModelIntervalVariable**, and **RadApp.ModelInterface.ModelVariables.ModelTextOutputVariable**.

4.0.29.1 Detailed Description

4.0.30 RadApp.ModelInterface.ModelVariables.ModelTextOutputVariable Class Reference

Output implementation

Inherits **RadApp.ModelInterface.ModelVariables.ModelOutputVariable**.

4.0.30.1 Detailed Description

4.0.31 RadApp.ModelInterface.ModelVariableProvider Class Reference

Model variable provider abstract class. Implement this to provide all the model variables instances necesarry in your model. Here is also where any remaining logic should be stored, because all the other classes are common for all contents. Provided methods can be overridden, if yoy wish to use other structure than dictionary for organising your model variable instances.

Inherits MonoBehaviour.

Public Member Functions

- abstract void **Initialise** ()

  Use this function to initialise the variables dictionary.
- virtual ModelVariable[] **GetAllVariables** ()

  Returns all modelVariable instances.
- virtual ModelVariable **GetVariable** (string name)

  Gets the model variable by its name
- virtual object **GetVariableValue** (string name)

  A convenience method that returns the variable value instead of its reference.

Protected Attributes

- Dictionary< string, ModelVariable > **variables**

  The variables dictionary. This strucutre is used because it faster than list, and more convenient than an array, when working with larger numbers of variables. You can use your any other implementation, as long as you override the necesary methods so that they provide the same interface for your implementation.

4.0.31.1 Detailed Description

4.0.31.2 Member Function Documentation

4.0.31.2.1 virtual ModelVariable [] RadApp.ModelInterface.ModelVariableProvider.Get↩ AllVariables ( ) `[virtual]`

Returns

All model variables.

4.0.31.2.2 virtual ModelVariable RadApp.ModelInterface.ModelVariableProvider.Get↩ Variable (string name) [virtual]

Returns

The model variable with given name

Parameters

name	Name of requested variable

4.0.31.2.3 virtual object RadApp.ModelInterface.ModelVariableProvider.GetVariableValue (string name) [virtual]

Only use when you are certain of the variable type.

Returns

The variable value as an object, which is necessary to provide any type of value.

Parameters

name	Name of the requested variable

4.0.31.2.4 abstract void RadApp.ModelInterface.ModelVariableProvider.Initialise ( ) [pure virtual]

4.0.31.3 Member Data Documentation

4.0.31.3.1 Dictionary<string, ModelVariable> RadApp.ModelInterface.ModelVariable↩ Provider.variables [protected]

4.0.32 RadApp.Controllers.Private.ParameterBarControler Class Reference

Class that manages the parameter bars, fills them with given variable visual components,

Inherits MonoBehaviour.

Public Member Functions

- void **HideBars** ()

  Hides both left and right parameter bar.
- void **ShowBars** ()

  Shows both the left and right parameter bar.
- void **SetName** (string name)

  Sets the name in headers of the bars.
- void **AddVariable** (ModelVariable var)

  Adds a new model variable.
- void **AddVariables** (**ModelVariableProvider** varsContainer)

Convenience method for adding whole arrays of model variables from the given variable provider.

- void **RemoveVariables** ()

  Removes all the model variables from the parameter bars.

- void **ResetVariables** ()

  Resets all input components to their default values.

4.0.32.1 Detailed Description

4.0.32.2 Member Function Documentation

4.0.32.2.1 void RadApp.Controllers.Private.ParameterBarControler.AddVariable (ModelVariable var)

Parameters

var	The variable to be added.

4.0.32.2.2 void RadApp.Controllers.Private.ParameterBarControler.AddVariables (**ModelVariableProvider** varsContainer)

Parameters

vars	The variable provider that the variables will be read from

4.0.32.2.3 void RadApp.Controllers.Private.ParameterBarControler.HideBars ( )

4.0.32.2.4 void RadApp.Controllers.Private.ParameterBarControler.RemoveVariables ( )

4.0.32.2.5 void RadApp.Controllers.Private.ParameterBarControler.ResetVariables ( )

4.0.32.2.6 void RadApp.Controllers.Private.ParameterBarControler.SetName (string name)

Parameters

name	New name to be shown

4.0.32.2.7 void RadApp.Controllers.Private.ParameterBarControler.ShowBars ( )

4.0.33 RadApp.Controllers.Private.SceneDisplayControler Class Reference

This class controls which content will be shown in the scene. It has access to the current content controler, holds stack of previously displayed contents and also does all the routines related to displaying new content. That includes toggle of the newx/prev buttons, action bar setting, main bar message and timers, and the side bar settings.

Inherits MonoBehaviour.

Public Member Functions

- void **ShowNextContent** ()

  Shows the content set as the next content property of current content (if there is any).
- void **ShowPrevContent** ()

  Shows the previous content in the navigation stack (if there is any).
- void **ShowContent** (GameObject content, int state)

  Displays given GameObject as scene content, and pushes the previous game content to the navigation stack. The given GameObject must have ContentControler component attached to it to be displayed.

4.0.33.1 Detailed Description

4.0.33.2 Member Function Documentation

4.0.33.2.1 void RadApp.Controllers.Private.SceneDisplayControler.ShowContent (GameObject content, int state)

Parameters

content	Content to be displayed
state	State parameter that will be passed to the displayed content.

4.0.33.2.2 void RadApp.Controllers.Private.SceneDisplayControler.ShowNextContent ( )

4.0.33.2.3 void RadApp.Controllers.Private.SceneDisplayControler.ShowPrevContent ( )

4.0.34 RadApp.Utils.Events.ScreenOrientationChangedEvent Class Reference

Event used in ScreenOrientationDetector, that is fired when a screen orientation change is detected.

Inherits UnityEvent< ScreenOrientation >.

4.0.34.1 Detailed Description

4.0.35 RadApp.Utils.Events.SelectChangedEvent Class Reference

Event used in the SlotSelectControler that is fired when the selection has changed.

Inherits UnityEvent< int >.

4.0.35.1 Detailed Description

4.0.36 RadApp.ModelInterface.SlotControlers.SlotInteractiveControler Class Reference

A placeholder class in case it is necesarry to manipulate all interactive controlers as one type, or extract commont functionality.

Inherits RadApp.ModelInterface.SlotControlers.SlotTextControler.

Inherited by RadApp.ModelInterface.SlotControlers.SlotBooleanControler, RadApp.Model↩ Interface.SlotControlers.SlotIntervalControler, and RadApp.ModelInterface.SlotControlers.↩ SlotSelectControler.

4.0.36.1 Detailed Description

4.0.37 RadApp.Utils.Graphs.StaticGraph Class Reference

Time graph is class that creates visual time-dependent graph of given variables. Use Draw↩ Value to draw the next value. Time resolution is not currently controlled by time, but by the frequency with which the DrawValue function os called (usually in an Update function of another script). If you want specific refresh rate, use timing function to controll the frequency externally (e.g. using MonoBehaviour.InvokeRepeating).

Inherits Object.

Public Member Functions

- void **DrawGraph** (StaticGraphDataSource dataSource, bool ownScale)

  Draws the graph from given data points. The lines are drawn between the points in array, respectively, so you should make sure that the data are correctly sorted.
- void **DrawPoint** (StaticGraphDataSource dataSource, bool ownScale)

  Draws a point given by the StaticGraphPoint[0] in the passed data source. Use the data source lineWidth property to specify the radius of the drawn point, and lineColor to specify its color.

4.0.37.1 Detailed Description

4.0.37.2 Member Function Documentation

4.0.37.2.1 void RadApp.Utils.Graphs.StaticGraph.DrawGraph (StaticGraphDataSource dataSource, bool ownScale)

Parameters

data	Data to be drawn.

4.0.37.2.2 void RadApp.Utils.Graphs.StaticGraph.DrawPoint (StaticGraphDataSource dataSource, bool ownScale)

Parameters

dataSource	Data source.

ownScale	If set to `true`, the dataSource will set its own scaling (based on the min/max-X/Y values). The graph scale will be used otherwise.

4.0.38 RadApp.Controllers.Private.StaticGraphControler Class Reference

This class is a wrapper for the StaticGraph utility, that creates a more convenient interface and provides setters of value labels and axes names.

Inherits MonoBehaviour.

Public Member Functions

- void **SetAxesNames** (string Xname, string Yname)

  Sets the axes names. Not to be confused with axes value labels, which are set in the Unity inspector on the object this controler is attached to.
- void **AddLinePlot** (StaticGraphDataSource dataSource)

  Adds a line plot created by lines between points in the given data source. The first added dataSource will be used to set the graph scale. Is is therewore advised to add the biggest data set first, or to match all the data scales before providing them here.
- void **AddPoint** (StaticGraphDataSource dataSource)

  Adds the first point in the given data source. The first added dataSource will be used to set the graph scale. Is is therewore advised to add the biggest data set first, or to match all the data scales before providing them here.
- void **RemovePoint** (StaticGraphDataSource dataSource)

  Removes the given dataSource from the points list and redraws the graph, so the point is actually removed from the visual component.
- void **RemoveAllPoints** ()

  Removes all points fromt the graph and redraws it.
- void **RemoveLine** (StaticGraphDataSource dataSource)

  Removes the give dataSource from the line list and redraws the graph, so the line is actually removed from the visual component.
- void **RemoveAllLines** ()

  Removes all lines and redraws the graph.
- void **RedrawAll** (StaticGraphDataSource scaleSource)

  Redraws the graph using the current line and point list. LInes are drawn first, so the points are drawn over them.

4.0.38.1 Detailed Description

4.0.38.2 Member Function Documentation

4.0.38.2.1 void RadApp.Controllers.Private.StaticGraphControler.AddLinePlot (StaticGraphDataSource dataSource)

Parameters

dataSource	Data source.

4.0.38.2.2 void RadApp.Controllers.Private.StaticGraphControler.AddPoint (StaticGraphDataSource dataSource)

See also

StaticGraph.DrawPoint

Parameters

dataSource	Data source.

4.0.38.2.3 void RadApp.Controllers.Private.StaticGraphControler.RedrawAll (StaticGraphDataSource scaleSource)

Parameters

scaleSource	The data source whose min a max values are used to set the graph scale.

4.0.38.2.4 void RadApp.Controllers.Private.StaticGraphControler.RemoveAllLines ( )

4.0.38.2.5 void RadApp.Controllers.Private.StaticGraphControler.RemoveAllPoints ( )

4.0.38.2.6 void RadApp.Controllers.Private.StaticGraphControler.RemoveLine (StaticGraphDataSource dataSource)

Parameters

dataSource	Data source.

4.0.38.2.7 void RadApp.Controllers.Private.StaticGraphControler.RemovePoint (StaticGraphDataSource dataSource)

Parameters

dataSource	Data source.

4.0.38.2.8 void RadApp.Controllers.Private.StaticGraphControler.SetAxesNames (string Xname, string Yname)

Parameters

XLabel	X label.
YLabel	Y label.

4.0.39 RadApp.Utils.StopWatch Class Reference

Record time until Stop function is called. Recording can then be returned or reseted with Start or Reset functions respectfuly. You have to call the StartRecording function after initialisation manually to start recording time.

Inherits MonoBehaviour.

Public Member Functions

- void **StartRecording** ()

  Start the time recording. Doesnt reset any recorded time.
- void **StopRecording** ()

  Stop the time recording. Keeps the recorded time.
- void **ResetRecording** ()

  Reset the recorded time. Does not interrupt recording, if the stopwatch is recording.
- bool **IsRecording** ()

  Checks whether the stop watch is currently recording.
- float **GetRecordedTime** ()

  Gets the recorded time in miliseconds.
- string **GetRecordedTimeString** ()

  Gets the recorded time as a formated string in format MIN:SEC:MS

4.0.39.1 Detailed Description

There is no interface for doing stuff when certain time is reached. You can use the **Timer** (p. 36) class for that, since it exposes an event that is invoked when it reaches zero. Stopwatch is here for monitoring purposes, e.g. creating an application log that would show how long was each content displayed.

4.0.39.2 Member Function Documentation

4.0.39.2.1 float RadApp.Utils.StopWatch.GetRecordedTime ( )

Returns

The recorded time.

4.0.39.2.2 string RadApp.Utils.StopWatch.GetRecordedTimeString ( )

Returns

The recorded time in formated string.

4.0.39.2.3 bool RadApp.Utils.StopWatch.IsRecording ( )

Returns

`true` if this stop watch is recording; otherwise, `false`.

4.0.39.2.4 void RadApp.Utils.StopWatch.ResetRecording ( )

4.0.39.2.5 void RadApp.Utils.StopWatch.StartRecording ( )

4.0.39.2.6 void RadApp.Utils.StopWatch.StopRecording ( )

4.0.40 RadApp.Utils.Events.SwipeDetectedEvent Class Reference

Event used in SwipeDetector, that is fired when a swipe across the screen is detected

Inherits UnityEvent< SwipeDirection.direction >.

4.0.40.1 Detailed Description

4.0.41 RadApp.Utils.Graphs.TimeGraph Class Reference

Time graph is class that creates visual time-dependent graph of given variables. Use Draw↩ Value to draw the next value. Time resolution is not currently controlled by time, but by the frequency with which the DrawValue function os called (usually in an Update function of another script). If you want specific refresh rate, use timing function to controll the frequency externally (e.g. using MonoBehaviour.InvokeRepeating).

Inherits Object.

4.0.41.1 Detailed Description

4.0.42 RadApp.Utils.Timer Class Reference

Timer (p. 36) class that counts down from preseted time in minues or seconds. Use the SetTime function to initialise the **Timer** (p. 36). You have to manually call StartCounting funtion to start the timer.

Inherits MonoBehaviour.

Public Member Functions

- void **StartCounting** ()

  Starts the counting. Does not reset the timer.
- void **StopCounting** ()

  Stops the counting. Does not reset the timer.
- void **SetTime** (int minutes, int seconds)

  Sets the timer to given minutes and seconds. Does not interrupt counting.
- float **GetRemainingTime** ()

Gets the remaining time in miliseconds.

- string **GetRemainingTimeString** ()

  Gets the remaining time as a formated string in format MIN:SEC:MS

4.0.42.1 Detailed Description

4.0.42.2 Member Function Documentation

4.0.42.2.1 float RadApp.Utils.Timer.GetRemainingTime ( )

Returns

The remaining time in seconds.

4.0.42.2.2 string RadApp.Utils.Timer.GetRemainingTimeString ( )

Returns

The remaining time in formated string.

4.0.42.2.3 void RadApp.Utils.Timer.SetTime (int minutes, int seconds)

Parameters

minutes	Minutes.
seconds	Seconds.

4.0.42.2.4 void RadApp.Utils.Timer.StartCounting ( )

4.0.42.2.5 void RadApp.Utils.Timer.StopCounting ( )

4.0.43 RadApp.Utils.Exceptions.VariableNotFoundException Class Reference

Variable not found exception is used ModelVariableProvider to be invoked when there has been a request for a variable, whose name does not exist in the local dictionary.

Inherits Exception.

4.0.43.1 Detailed Description

4.0.44 RadApp.Utils.Exceptions.VariableProviderNotAttachedException Class Reference

Variable provider not attached exception, used during the content manipulation

Inherits Exception.

4.0.44.1 Detailed Description

4.0.45 RadApp.Utils.Exceptions.WrongVariableTypeException Class Reference

Wrong variable type exception is used in ModelVariable and SlotControlers to be invoked when a wrong value type or event has been accessed.

Inherits Exception.

4.0.45.1 Detailed Description

Appendix D

CD contents

There is a CD attached with the printed copies of this thesis, and a ZIP archive is attached with the electronic version of this thesis. Both the cd and the archive contain the following folders:

D.1 AcidBaseApp

This folder contains the example application builds. There are two archives. One with the Android .apk file, that can be deployed on an Android device, and another with a windows standalone executable and the required data.

D.2 Documentation

Here, the LaTeX files, figures and pictures, that were used to create this document, are located.

D.3 Radapp

This folder contains the entire Unity project with the RadApp framework. Navigating to this folder in Unity will open the environment described in this thesis (the ZIP archive has to be extracted first).

— *FIN* —