Ing. Roman Zeleznik BSc

Efficient implementation of virtual physics experiments for web based interactive courses at the MIT

Report - Marshall Plan Foundation

Graz University of Technology

Institute for Information Systems and Computer Media - IICM
Head: Univ.-Prof. Dipl-Ing. Dr.techn. Frank Kappe

Massachusetts Institute of Technology

Center for Educational Computing Initiatives - CECI
Head: Professor John Belcher

Supervisor: Univ.-Doz. Dipl-Ing. Dr.techn. Christian Guetl
Co-Supervisor: Professor John Belcher

Boston, July 2014

Contents

Abstract

Teaching freshman MIT students physics is a challenging task, as every MIT student has to go through this courses no matter what topics they are studying. As the student are hence not very interested as well as the rest of their study is very time consuming it is easy to understand why the students do not like this courses. Trying to motivate the students to learn the physics stuff the courses were ported to so called e-learning courses as evaluations in the past have shown that this new form of education can increase students motivation as well as simplify learning. Especially interactive simulations help student understand the taught topics.

Therefore in 2005 *Technology Enabled Active Learning* (TEAL) was introduced. Therefore a special TEAL classroom was developed which combined traditional teaching with interactive elements like videos, simulations and physics experiments.

As technology got faster and better and the Internet got a permanent part of our life a new e-learning platform called edX was created. It is an online platform for many different courses. Two of them are the basic physics courses 8.01x and 8.02x which are held in the TEAL classrooms.

For giving students a easier and more flexible access to the physics experiments 7 of the original TEAL simulations where ported to edX using JavaScript. This simulations give the students the possibility of working whenever they want and as often they want and should increase students motivation and interest in the physics stuff.

Unfortunately evaluations of the courses have shown that students did not or just very short make use of the simulations. Therefore the wish was born to make them interact with edX to give the course creator the possibility of asking students questions which have to be answered by using the simulations.

In this master thesis we created an interface between the existing TEAL simulations and the edX platform. Furthermore we extended 3 of the simulations which now are used for answering questions by using them.

Acknowledgement

I want to thank so many people for making this master thesis as incredible as it is. I have to thank all of you for your support and guidance through my work. In the six months I wrote this master thesis I had the opportunity to work with some of the best scientists at one of the best universities.

First I want to thank all the people from Boston, especially the CECI team. My thanks go to John for making all this possible, for showing me a whole new form of education and a new direction in which I want to steer my future work and for introducing me to so many wonderful people. Thanks to Phil for all your support and help with all the technical stuff. Thanks to Kirky and Maria for all the administrative support. At this point a special thanks to Kirky for all the help before my trip, answering my thousand mails and your help getting my visa! Thanks to Luciano for having lunch so many times with me. Thanks to Kimberle for bringing Kyla into the office. Furthermore I want to thank Saif for the work on the edX course with me. And of course thanks to Ivan for all your help and for the bike. Thanks also to Mark for all the wonderful stories about the MIT, Boston and the US and for the many recommendations for our weekends and vacation.

Thanks all of you!

And I want to thank every one from the Graz University of Technology. Especially Christian for making this all possible and for your continuous help and support through all the six month in Boston and the time back in Graz. Thanks also to Johanna for all the help getting the visa and also the technical help through my work.

I want to thank the Marshall Plan foundation for the financial support which made this journey possible!

From my home town Graz I want to thank Karli and Eveline for having an eye on our apartment and all the other things you helped me during my trip!

Finally I want to thank my love and life Oana. Words can cot describe how much I owe you!

1. Introduction

At the Massachusetts Institute of Technology (MIT) every undergraduate student has to take two semesters of basic physics, no matter which field of studying they are going to take. These courses are tough and require a lot of work and hence not very favored by the students. Since 2005 the physics courses are held in so called *Technology-Enabled Active Learning* (TEAL) classes. TEAL is a method for giving the students, among other things, the possibility to work on their own with different physics experiments. This changed the traditional and passive way of teaching to an interactive way where students can test their understanding. There were different reasons for the TEAL project [7]. One was to decrease the number of students failing the courses. Another was to make studying easier and hopefully to make students remember the learned material for the rest of their life. It was also made to motivate the students to learn physics. During one of the first meetings, Prof. Belcher explained that he wanted the students to not hate the courses any more. And if they just dislike it instead of hating it, he would be happy.

With technology getting faster and better TEAL-simmulations (TEALsim) was created. It was a stand alone software including several virtual TEAL simulations which where used in TEAL classes but could be used by every student at home also.

The last evolutionary step was to use the e-learning platform edX for teaching the physics courses. These courses included the whole course material as well as 7 ported virtual TEAL simulations. Furthermore online questionnaires with instant feedback and online homeworks are part of these edX courses. Evaluations of further semesters have shown that students really liked the edX courses [12]. Most important for them was the immediately feedback they received for online questions. Past works have also shown that interactive simulations can improve students learning [5]. That was also the reason why the TEAL simulations where ported to edX. Evaluations of the physics courses have shown that students did not use them or used them insufficiently. Prof. Belcher wanted to extend the existing 7 TEAL simulations with the possibility of asking students questions during the online course which have to be answered by using them. This would on the one hand make the students using them, on the other hand it would give them the fast feedback they liked most from the edX courses.

I. Ceraj did something similar at various courses, for example 7.QBWx, 8.EFTx and others. They also contain interactive experiments which were used to answer questions and solve tasks. But differently to this courses, where he created the edX part as well as the JavaScript part, our courses were created by two different people. One was responsible for the course content, which was written in XML and python, the other for the simulations, which were written in Java and JavaScript. So we had not only to extend the existing simulations, we also had to create an interface between them and the course content to keep the strict separation.

Then we had to think about how to change the existing simulations for giving the lecturers the possibility of asking useful questions which increase students motivation for using them. Once all requirements where set we began to work on the implementation. J.M Claus from edX recommended the using of jsinput, an edX function made for interacting with JavaScript implementations like our simulations. Here we had to deal with a peculiarity of the existing simulations as they where written in Java and compiled with the *Google Web Toolkit* (GWT) to JavaScript Code. This made a direct use of jsinput with our simulations impossible. Once the programming work and implementation was done we had to present it to the physics education stuff which consisted of brilliant, well experienced and very smart MIT lecturers. Here we realized that our work was not just to create a working interactive simulation with an stable interface. Each decision was questioned, every tiny detail had to be correct and everything had to be perfect to fulfill the high standards of the MIT.

1.1. Problem Definition

In order to solve the problems described above two main objectives were defined.

Objective One

Extend one of the existing virtual TEAL simulations which where ported to edX for giving the course creator the possibility of asking students questions which have to be answered by using them. This includes on the one site to setup a virtual edX server locally for creating and testing an prototype course and furthermore understanding and editing an existing course. On the other hand understand the implementation of the existing simulations. After that we had to think about how to extend the simulations for asking useful questions. This was an interesting process which included many different people from education as well as programmers. We had to combine the wishes of lecturers with the possibilities the programmers had and take into account the available time period.

Objective Two

Create an interface between the extended TEAL simulation and the course which embeds it. This was a very important aspect as the main course was created by educational stuff who have no knowledge of programming but the simulations where created by programmers which where not familiar with education. While this circumstance was very reasonable for creating the best physics course for a great education it also led to the requirement of an interface between simulation and online course which ensures none of the two creators has to work in the others field.

1.2. Structure of the following work

This master thesis is divided into three main parts. The first section describes the background and basics of this work. It encompasses the chapters 2, 3 and 4. Part two is about the implementation and our results, containing chapters 5 and 6. The third and last part includes a small evaluation in chapter 7 as well as an outlook in chapter 8.

In chapter 2 we explain the terms *e-learning*, *MOOCs* and *simulation* and their application in education. Furthermore we describe the *Technology-enabled active learning* teaching format and the conversion to TEALsim with the virtual simulations, which we want to extend in this work. We give an overview of edX, an online e-learning platform for which the physics courses and hence our simulations were created and we show the structure of such an online course and the programming languages which were used. Finally we give an introduction to GWT, the *Google Web Toolkit* and how an edX course can be extended using it. Chapter 3 illustrates the problem definition. Therefore we get a little more into the relation of TEALsim and edX. After that we specify the individual points we want to accomplish with this work. In chapter 4 we write about related work to this master thesis. We show the innovation of our work and consequently why there was so less related work we could build on. Afterwards two concepts which we thought we can use for our work are explained and why they turned out to be incompatible for our requirements.

Chapter 5 refers to the whole implementation of our work. It shows the underlying technologies, which problems we ran into and how we solved them. For better understanding we show a text course which we created for testing the new mechanisms. At the end we present a float chart which explains the whole structure of our simulations and its interaction with edX. In chapter 6 we show the final results of our work, the three converted simulations. We describe how we extended the existing simulations, which changes had to be done, where the problems were and finally the implementation and underlying mechanism of our new simulations. As the three new simulations are all different we made an own sub-chapter for each one where we show the ideas and possibilities how they could be used in future courses.

Chapter 7 is sadly a little shorter then we hoped it would be. During our work on the simulations it seemed they will be used in the following term by the edX course 8.02x which would have given us an evaluation of about 150 MIT students for this thesis. But different reasons lead to a delay in integrating them into the course and so they will be used after this thesis is written. Thus we made a evaluation with colleagues and student assistants which is unfortunately with less people. In the last chapter 8 we show a short outlook on how our simulations will be used in online courses. Furthermore we describe how our work will be continued and where it may lead to and which modifications to edX and its functions would have to be done.

2. Background and Technologies

Electronic learning (e-learning) and *Massive Open Online Courses* (MOOCs) are both relatively new learning methods, especially MOOCs. They became very famous during the last years but have also changed enormous during that time. Used techniques are very different and hence results and experiences too. Therefore we want to give only a short general description of this two terms and present then an evaluation of the physics course 8.02x to describe the benefits and disadvantages of their application specifically in this course.

2.1. E-learning and MOOCs

With the increasing use of computers and the spread of network technologies and the Internet, teaching is continuously moving from the traditional classroom learning to electronic-learning (e-learning) as described in the work of Zhang et al. [14]. He defines e-learning as every form of learning where the course material is delivered electronically via the Internet. Therefor different systems can be used:

- E-mails
- Newsgroups
- Knowledge boards
- Online databases
- Websites

For us the last point, *Websites*, is most interesting. In the near past, especially since the rise of JavaScript, websites became more and more the preferred way of e-learning. This progress was supported with the upcome of HTML5 which, although it is officially not a released standard yet, is supported by every modern browser by today (2014). The benefits of e-learning are numerous: It is completely time and location flexible. The latter leads to the next advantage, cost and time saving. Up to 40% of in-person corporate learning are spend for traveling [14]. Furthermore the students gets unlimited access to learning materials so they can reuse it as much as they need to understand a specific topic or also skip parts they maybe already know. Zhang et al. also found out, that no significant difference of effectiveness of e-learning and traditional learning.

In 2011 a new learning method was introduced: Massive open online courses (MOOCs) which is topic of the work of Stephen Brown [4]. MOOCs are, as their name says, online courses which are available over the Internet for a huge amount of learners. Today most well known universities like MIT, Harvard, Stanford University and many more are offering such online courses but also companies are using it for training their employees and for support services. The most powerful advantage of e-learning is feedback. This can be obtained in two ways: First, by analyzing the users conduct. How long are students working on which task? How often is a question answered right or wrong? How many students are passing the course? And many more. With all this information we are able to adjust the course for the next iteration. We can identify which topics are maybe too less explained or which homework was too complicated. The second way is by asking students questions at the end or maybe also at the middle of the course. S. Rayyan et al. did so for the current term (autumn 2014) [12]. Some results are shown in section 2.1. A benefit of MOOCs is that with only a view and short questions, which take every student just a few minutes, we can accomplish a huge feedback which can be used for improving the course.

Evaluation of prior courses

The magnificent positive feedback of prior terms of 8.01x and 8.02x was the impulse for this master thesis as the wish was merged to improve and extend these courses. Therefor we want to show some results of the mid-semester evaluation of the physics course 8.02x 2014, see figure 2.1. Very interesting is the result for *Checkable answers on MITx for written Pset problems* as the goal of this work is to extend exactly the referred problem set. Furthermore we want to show the result of two specific questions in figure 2.2. The evaluation shows that MOOCs is working great for 8.02x and students really like the way they are taught. We will get back to this in section 7.

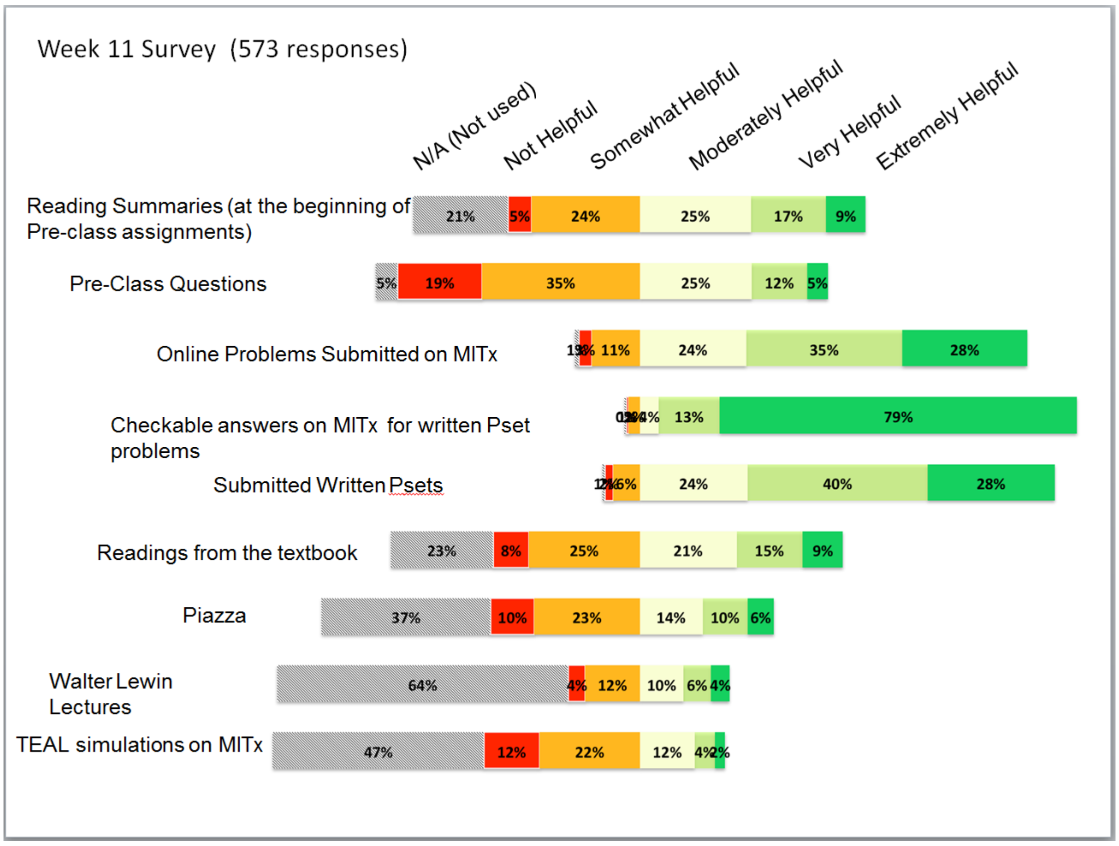
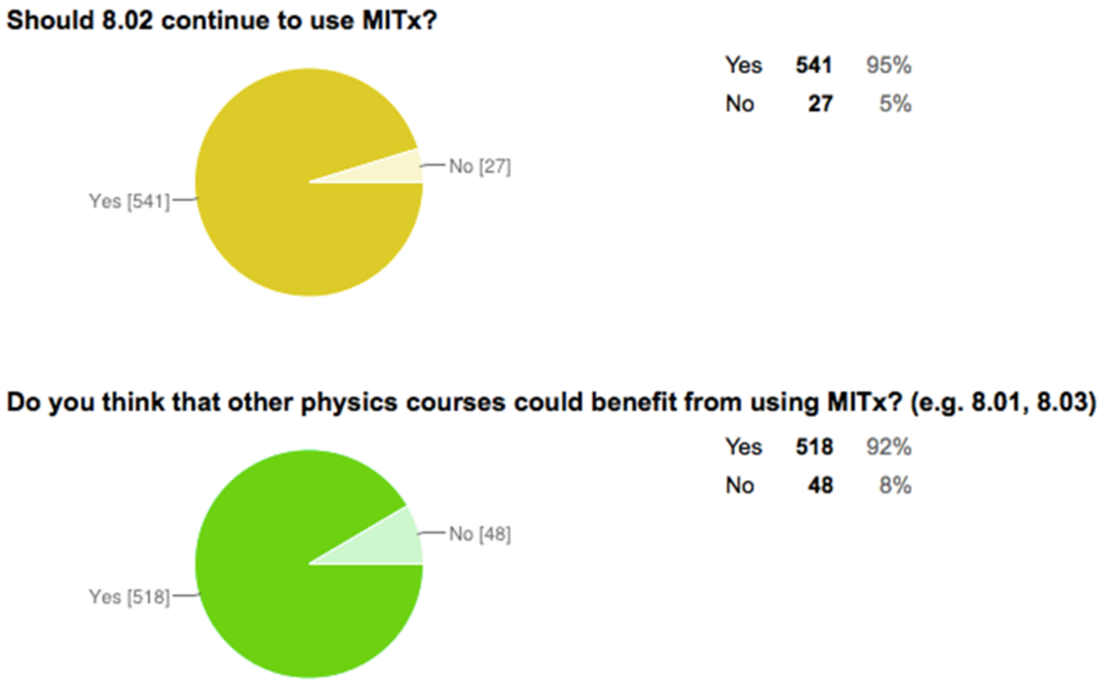


Figure 2.1.: Mid-Semester evaluation of 8.02x, spring 2014 [12]



Should 8.02 continue to use MITx?

Yes	**541**	95%
No	**27**	5%

Do you think that other physics courses could benefit from using MITx? (e.g. 8.01, 8.03)

Yes	**518**	92%
No	**48**	8%

Figure 2.2.: Mid-Semester evaluation of 8.02x, spring 2014 [12]

2.2. Simulations

As Gibbons et al. outlines in his work [9], a simulation simulates something. That can be on the one hand things that really exists, for example a physics experiment demonstrating the gravity. On the other hand a simulation can be used to explore and present things that we do not know if and how they exists like a black whole in the universe. Therefore a simulation creates a set of things and build the relationships between them which can be defined by simple rules to very complex mathematical expressions. After a simulation is started a user can perform changes to the state or values of the simulation and observes the consequences.

Simulations open up the possibility of changing things, that can not be changed in real life, for example, again, the gravity on earth. They can also be used for experiments which would be too expensive or simply to difficult in real life. Furthermore simulations are predestined for dangerous experiments that would risk human life.

Since many decades simulations were used in education to help students understand processes and coherences of various topics as described by Clayton et al. in his work [5]. Many students have troubles understanding the relevance of the topics they were taught or were not able to apply the learned material to real life problems. Here simulations can demonstrate applications of learned subject matter to simplify learning and understanding. Simulations can also help motivate students learning very complex things or topics in which they were not so interested in. The higher interest leads to a deeper understanding and furthermore to learning more in less time [5].

Simulations, games and gamification

At this point we want to step a little deeper into the terms *simulation* and *game* and their difference. For us, our virtual physics experiments where always simulations. But when we had some colleagues shown them we frequently got the answer *"That's funny, let me play with it!"*. It seemed we accidentally *gamified* our simulations. Accidentally, because we never actively tried to do so. But it was the reason to step into this matter during this master thesis. We will discuss the expression gamification a little later in this section and the impact of it on our results in chapter 6.

We want to look into the differences about games and simulations. When we searched the literature about simulations in education we consistently came across the term *game*. Very often the terms *simulation* and *game* are used in the same context. Interestingly, as shown by Crookall at al. [6], the term *game* is used to refer to the term *simulation* but never in the other way around. He defined the differences between games and simulations. As many features are same within both of them there are two man differences.

1. A game does not represent a real world system. Of course a game can be based on real world situations and environments, like for example ego shooters, but they are not supposed to be realistic. A simulation in contrast is always made to be as realistic as necessary as it should reflect real world behavior. A very interesting example here is the game Chess. It was initially made as a simulation but over years the reference to real life got lost as the world has changed and nowadays it is a definitely a game. A game has it's own world with it's own rules and rights while a simulation always tries to project the real world. That means a game is a real system but a simulation is just a meta system.
2. The error cost. An error during a game can lead to real life costs like for example losing money by playing Poker. The costs of losing a game can

be very small, maybe only a wounded pride. In contrast, a simulation can not be lost.

With this knowledge Crookall at al. realized that we can create a simulation of a game but we can not create a game of a simulation[6].

Now that we know about simulations, how they are used to improve education and where the differences to games are, we want to extend our knowledge about games and their place in education.

Games in education

Pim van de Pavoordt has shown in his work [13] that games can improve education. Main reasons are on the one hand fun while playing games. On the other hand he described the environment in which young people today are growing up. The students of today use computers, laptops, smartphones and the Internet since they where very young. They are very familiar with this techniques and thus they feel comfortable with them. This fact leads to more fun during learning and hence to higher engaging. That does not mean that games or simulations are the new and ultimate way of teaching, they are rather an extension of traditional methods. For this purpose already existing games can be used as well as new games specific made for education. A very famous game created for education is *Supercharged!* by Microsoft and MIT iCampus. It is *"an abstract world of electric charges, electric fields, magnetic fields, charged spheres - all of the basics of a physics textbook, but come to life in 3D"* for teaching physics at MIT [10]. Evaluations have shown that students who were playing Supercharged! performed about 20% better on the tests then students who were taught with traditional methods [13]. This facts lead to thoughts about how to increase the use of games in education as well as how to teach different topics through games which then leaded to the new expression *gamification*.

Gamification

As mentioned earlier in this chapter, during our work we were confronted with the fact that our simulations where called games. It seemed we unintentionally gamified our simulations. Muntean et al. describes in her work [11] gamification as the use of game elements in non-game applications for raising motivation and engagement of students. This should lead to having fun using the gamified applications and furthermore in spending more time with them. Hence Muntean et al. expect better results on final tests from students which where learning using gamified applications.

2.3. TEAL and TEALsim

As described in [7], teaching freshmen physics is a challenging task. The teaching material should motivate students to learn, as well as to understand the the things they are learning. *Technology-enabled active learning* (TEAL) is a teaching format that merges lectures, simulations and hands-on desktop experiments to create a rich collaborative learning experience. TEAL is used for almost all MIT introductory physics instruction. The TEAL classes feature:

1. Collaborative learning - students working during class in small groups with shared laptop computers
2. Desktop experiments with data acquisition links to laptops
3. Media-rich visualizations and simulations delivered via laptops and the Internet
4. Personal response systems that stimulate interaction between students and lecturers

A sketch of a TEAL classroom can be senn in figure 2.3.

The course design is based on the following premises:

1. Interaction between teacher and student is an important factor in promoting learning
2. Interaction among students is another
3. Active learning is better than passive learning
4. Hands-on experience with the phenomena under study is crucial

Even though traditional e-learning systems bring many advantages such as flexible learning possibilities, internationalization and cost reduction, STEM (Science, Technology, Engineering, and Mathematics) education is a challenging task. Educational principles based on constructivism can help improve the students' understanding. In particular, learning theories such as TEAL, based on interactive engagement activities are proven to improve the students' conceptual understanding of abstract domains such as physics. However, not every institution or university can apply this method due to the high implementation costs. In addition, the TEAL approach is not designed to enhance distance education scenarios. Traditional e-learning approaches do not seem to fulfill

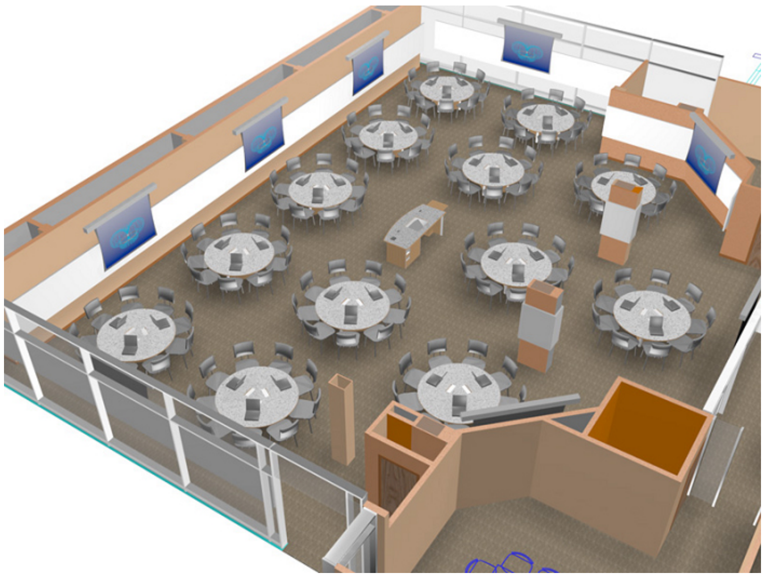


Figure 2.3.: A TEAL class room [7].

TEAL's requirements because of a lack of interactivity, motivational aspects and communicative and collaborative potential.

2.4. EdX - An online learning platform

EdX is a non-profit platform that provides free online classes and so called MOOCs (massive open online courses). It was created by the Massachusetts Institute of Technology (MIT) and Harvard and launched in May 2012. EdX is open Source and the code can be found on GitHub.

By today (March 2014) there are 32 edX charter members and 12 edX members offering 157 online courses. Famous examples of members and charter members are Massachusetts Institute of Technology, Harvard University, Berkeley University of California, ETH Zürich, Caltech, University of Toronto and University of Hong Kong.

The online courses are located in a wide variety of subjects including physics, law, history, science, engineering, business, social sciences, computer science, public health and artificial intelligence. The offered courses are free and open, what means everyone all over the world is able to join them.

EdX was created with three goals in mind:

1. Open up access to quality education globally
2. Improve on campus education
3. Conduct research to understand more precisely how students learn

2.5. EdX and the mix of HTML, Python, JavaScript and Latex

The edX platform allows the use of different programming languages for creating an online course. Generally said, when an edX course is opened in a browser everything shown is always HTML code maybe including some JavaScript code. This does not mean, that the whole course is written in HTML and JavaScript. As we will describe in the next section 2.6 there are different ways of creating an online course. The method we used included using XML and Python. XML is used for the whole course content like all the text as well as text-boxes or check-boxes for the online homeworks and also for hints and shown solutions of questions. Python is used for all the things behind the visible course site. For example for creating random variables, different calculations and computations as well as evaluating given answers. And the last language we used is latex. Normally latex is used for writing books, papers or journals but it can be also used for creating edX courses using MIT's latex2edx compiler.

2.6. Main structure of an edX online course

Certainly the goal of this thesis is not to describe how to create a whole edX course. However, for understanding how to embed the physics simulations in the edX course it is important to know at least the main structure of such a course.

Basically there are three different ways of creating an edX course:

1. Using edX studio: Studio is a browser based course management system see *https://studio.edx.org/*. It provides everything needed for creating an online course including an online editor, calendar, easy drag and drop tools and many more.
2. latex2edx: With this tool a whole course as well as individual problems can be written in latex. The latex source is then converted to edX XML code. This method was implemented because of the common use of latex

at universities and the well-known highly suitable mathematical content of latex. This method is used by the people at MIT for whom this thesis is written.

3. Using a plain editor: This method is of course the most complicated way to create an edX course but also gives the most capabilities and for programmers it is the most familiar way. Furthermore it is often used for editing existing courses where other tools meet their limits. This was the way the work was done in this master thesis.

Regardless which of this methods is used for creating and testing a course, a test environment is needed. Therefore edX offers a test server. A detailed manual can be found at: *http://people.csail.mit.edu/ichuang/edx/*. We used Oracle-VirtualBox for running a virtual Linux machine together with Vagrant for setting up the development environment. EdX offers a *.box* file which includes the Ubuntu Linux distribution with edX running and everything that is needed therefore. The edX test system can then be reached using a browser calling *http://192.168.42.2*. On this test environment a user is precreated which can be used for logging into the system and sign up for the created courses. This can be done by using the email *xadmin@mitxvm.local* and password *xadmin*. Of course more different users can be created. For more details see the manual linked above.

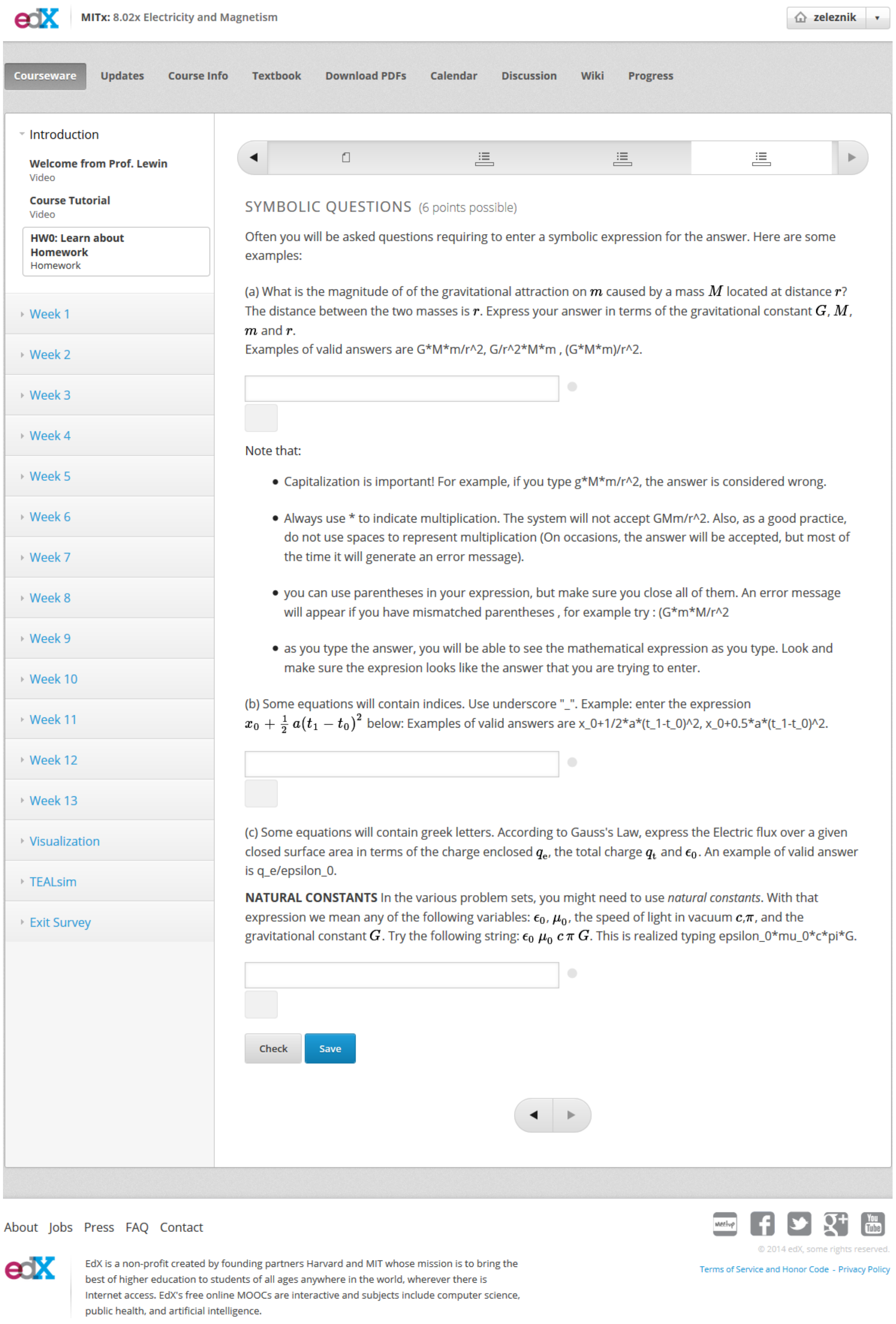
2.6.1. A basically setup of an edx course

An example of an edX course can be seen in figure 2.4. It shows the physics course 8.02x with a site for an interactive homework opened.

Every course contains of an horizontal top menu with links to general course dates and an vertical left menu which includes the course material divided in weeks. In the top menu two sections are interesting for our work:

- Courseware: This tab contains the main online course including Pre-Class assignments, online problems, forms for hand written homework, problem solving examples, visualization and other course stuff.
- Instructor (only visible if logged in as admin): This is an important section for creating a course. Here one can perform a *Reload course from XML files*. This has to be done if the course source XML files were changed.

In the left menu we will just need the homework links which will point to our simulations.



SYMBOLIC QUESTIONS (6 points possible)

Often you will be asked questions requiring to enter a symbolic expression for the answer. Here are some examples:

(a) What is the magnitude of of the gravitational attraction on m caused by a mass M located at distance r? The distance between the two masses is r. Express your answer in terms of the gravitational constant G, M, m and r.
Examples of valid answers are G*M*m/r^2, G/r^2*M*m , (G*M*m)/r^2.

Note that:

- Capitalization is important! For example, if you type g*M*m/r^2, the answer is considered wrong.
- Always use * to indicate multiplication. The system will not accept GMm/r^2. Also, as a good practice, do not use spaces to represent multiplication (On occasions, the answer will be accepted, but most of the time it will generate an error message).
- you can use parentheses in your expression, but make sure you close all of them. An error message will appear if you have mismatched parentheses , for example try : (G*m*M/r^2
- as you type the answer, you will be able to see the mathematical expression as you type. Look and make sure the expresion looks like the answer that you are trying to enter.

(b) Some equations will contain indices. Use underscore "_". Example: enter the expression $x_0 + \frac{1}{2}\,a(t_1 - t_0)^2$ below: Examples of valid answers are x_0+1/2*a*(t_1-t_0)^2, x_0+0.5*a*(t_1-t_0)^2.

(c) Some equations will contain greek letters. According to Gauss's Law, express the Electric flux over a given closed surface area in terms of the charge enclosed q_e, the total charge q_t and ϵ_0. An example of valid answer is q_e/epsilon_0.

NATURAL CONSTANTS In the various problem sets, you might need to use *natural constants*. With that expression we mean any of the following variables: ϵ_0, μ_0, the speed of light in vacuum c,π, and the gravitational constant G. Try the following string: $\epsilon_0\ \mu_0\ c\,\pi\ G$. This is realized typing epsilon_0*mu_0*c*pi*G.

EdX is a non-profit created by founding partners Harvard and MIT whose mission is to bring the best of higher education to students of all ages anywhere in the world, wherever there is Internet access. EdX's free online MOOCs are interactive and subjects include computer science, public health, and artificial intelligence.

© 2014 edX, some rights reserved.

Terms of Service and Honor Code - Privacy Policy

Figure 2.4.: An example for an edX course with an interactive homework opened.

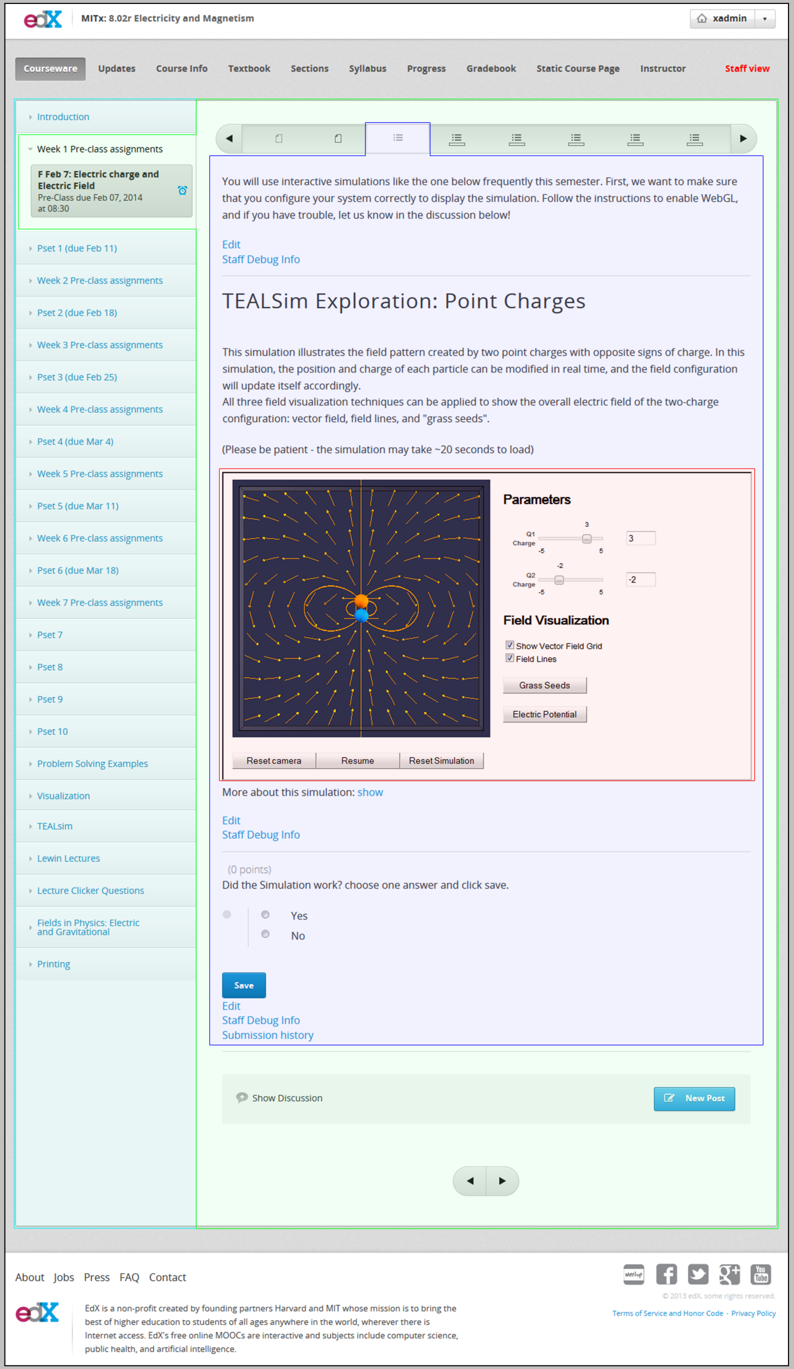
2.6.2. The setup of the courses 8.02x

This section is important for understanding how to extend standard edX problems for working with GWT compiled HTML pages. We describe the setup on the pre-class assignment week one site 3 of the MITx course 8.02 of spring 2014. As we will describe in chapter 7 the experiments written in GWT will be compiled in one single HTML file. For embedding we have to use an iframe which is an HTML5 compliant tag. Embedded HTML files should be located in the static/html folder. This is the place for non edX created and complete and static HTML files that are included in the dynamically created edX course.

The HTML file is included by hand written source code with an iframe tag:

```
<iframe id="teal-iframe"
src="/static/html/gwt-teal/PCharges2.html?size=400"
width="818"
height="475"
scrolling="no"
frameborder="1" />
```

The benefit of this method is that the course page is dynamically created by edX using XML. As seen in 2.6.2 a whole problem set can be created including questions, check-boxes, etc., only the experiment itself is external HTML code. Here the main problem and limitations of edX can be seen: There is no way of interaction between edX and the GWT created simulation.



You will use interactive simulations like the one below frequently this semester. First, we want to make sure that you configure your system correctly to display the simulation. Follow the instructions to enable WebGL, and if you have trouble, let us know in the discussion below!

Edit
Staff Debug Info

TEALSim Exploration: Point Charges

This simulation illustrates the field pattern created by two point charges with opposite signs of charge. In this simulation, the position and charge of each particle can be modified in real time, and the field configuration will update itself accordingly.
All three field visualization techniques can be applied to show the overall electric field of the two-charge configuration: vector field, field lines, and "grass seeds".

(Please be patient - the simulation may take ~20 seconds to load)

More about this simulation: show

Edit
Staff Debug Info

(0 points)
Did the Simulation work? choose one answer and click save.

Yes
No

Edit
Staff Debug Info
Submission history

EdX is a non-profit created by founding partners Harvard and MIT whose mission is to bring the best of higher education to students of all ages anywhere in the world, wherever there is Internet access. EdX's free online MOOCs are interactive and subjects include computer science, public health, and artificial intelligence.

© 2013 edX, some rights reserved.
Terms of Service and Honor Code - Privacy Policy

Remark: The segmentation of the edX course 8.02x in separate files

The physics course, for which our simulations are written, are separated in several files. At the beginning the file structure can be confusing wherefore we show here the file hierarchy of a course page with an example simulation. The colored areas are defined by the following file names with the same coloring.

1. **\802r-TTh \course.xml**
   This file only holds a link to the next file, *2014_Spring.xml*. It is the base file which prompts edX to create the main course page with the horizontal menu at the top of the page and all the administrative stuff.
2. **\802r-TTh \course \2014_Spring.xml**
   With this file the chapters of the course are created. At our example course 8.02x two chapters spans a week. One for pre-class assignments and one for the weekly homeworks.
3. **\802r-TTh \chapter \pcq \W01pcq.xml**
   Here the second horizontal menu beneath the first one is created. Every pre-class assignment or homework is divided into smaller parts which can be reached over this menu.
4. **\802r-TTh \vertical \802r \W01_D1_q1.xml**
   This is the file where the actual course data is written. It contains explanations, questions, hints and many more. This is also the place, where our simulations are included.
5. **\802r-TTh \static \html \gwt-teal \PCharges2.html**
   This is the HTML file with the simulation. It contains everything thats needed for running the simulation. It is created from GWT and compiled from the Java code. This simulation.html file is fully executable on it's own. That means it is just embedded in the edX course.

2.7. GWT - Extending edX with a new programming language

The Google Web Toolkit (GWT) is a toolkit that allows writing code in Java and compiling it to JavaScript. The main difference to other frameworks is that with GWT we are able to write server side as well as client side code in Java. For embedding our virtual experiments in online courses we depend on JavaScript, but writing such big applications directly in JavaScript is difficult and complex since it was not developed to implement big projects. Because of that, there exist almost no kind of debugging system for JavaScript. That is a big advantage of using GWT, as we are able to write our code in Java using our well known Java environment and test it with a powerful debugging system beneath it. Furthermore, the existing TEALsim experiments are written in Java. Unfortunately, this does not mean that we can take the Java written experiments and compile them straight forward to JavaScript using GWT. The TEALsim code contains a large number of external libraries that are not supported by GWT. At least we were able to use parts of the already existing code and for the parts we had to rewrite we could use Java which we were way more familiar with than with JavaScript. Another big advantage of GWT is that if we need JavaScript Code that is not available in GWT we can weave our Java code with JavaScript Code. This is important for example for accessing browser variables. Furthermore, GWT provides a great number of templates and libraries, such as html forms and input textboxes.

The GWT compiler

The main function of the GWT compiler is compiling Java code to JavaScript. While doing that it analyzes the written code and optimizes and minimizes it. The created code is then as compact as possible in order to keep data that has to be sent over the Internet as less as possible. The compiler also runs so called generators which are part of GWT and create code at compilation time. A generator replaces specific code, such as templates or other simplifications, and replaces it with new generated and working code. This decreases the amount of code that has to be written by the programmer. There exist a large number of generators which can furthermore also be own written. The compiler will call itself the right generator depending on the code. Another important task of the compiler is the generation of different versions of JavaScript for different browsers and therefor also a bootstrap loader. The bootstrap loader will be running on the client side and recall the right version for the client's browser.

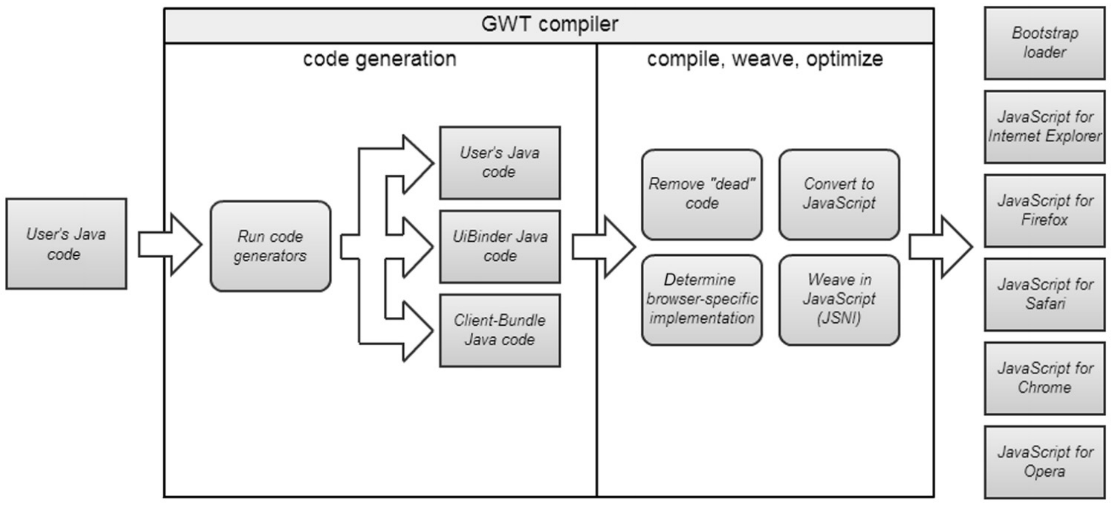


Figure 2.5.: The GWT Compiler in Detail [3]

This method is called *deferred binding* and allows the programmer take into account differences of how individual browsers are handling specific JavaScript code. An illustration of the GWT compiler can be seen in figure 2.5. How the generated code will be shown to a user can be seen in 2.6. A user enters a URL in the browser's address field. The browser and the server then interact with each other to create the appropriate application for the user and its system.

Advantages of GWT

There are many advantages of GWT especially the fact of being able to use Java, a high level object-oriented programming language, instead of JavaScript. Moreover GWT is perfectly adjusted for communicating between client and server which we needed to communicate between an iframe and its parent site, which was written using a mix of XML, python, HTML and JavaScript. In conclusion there is no direct access from GWT to edX variables and vice versa.

2.8. Summary

In this section we describe what e-learning, MOOCs and simulations are, how they are used for education and what their advantages and disadvantages are.

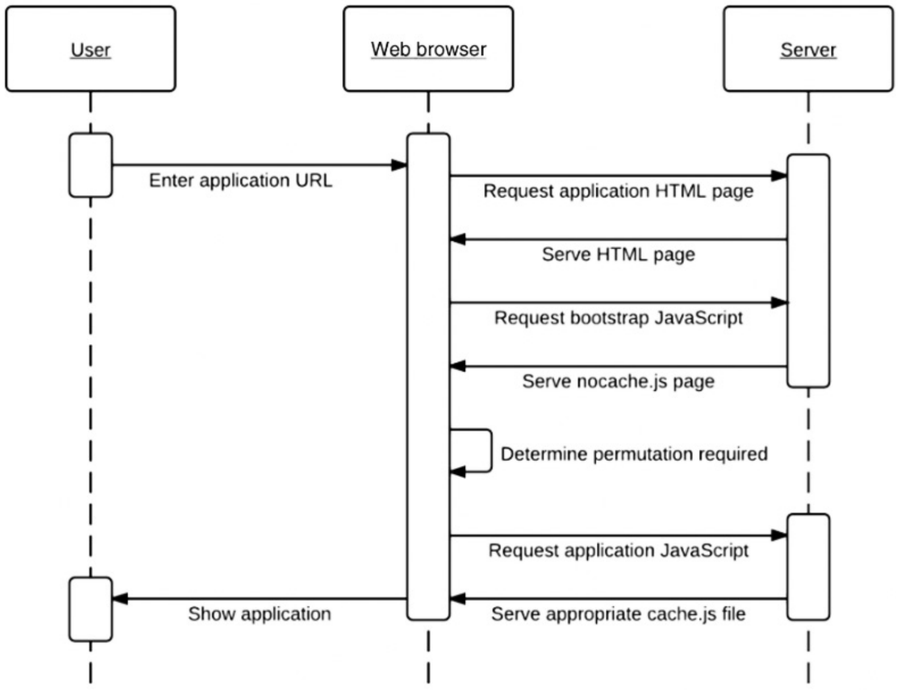

User
Web browser
Server
Enter application URL
Request application HTML page
Serve HTML page
Request bootstrap JavaScript
Serve nocache.js page
Determine permutation required
Request application JavaScript
Show application
Serve appropriate cache.js file

Figure 2.6.: Starting a GWT application [3]

With the mid-semester evaluation of the physics edX course 8.02x 2014 we show specific benefits that are reached by using these techniques at the MIT. Subsequently we describe the origins of the virtual physics experiments in TEAL and TEALsim followed by the online learning platform edX and how these simulations can be embedded in edX courses. We discuss how a course can be build and written and which techniques exist. At the end we show how we extended the existing course by the programming language Java which is compiled with GWT to JavaScript.

3. Problem Definition

During the first semesters of each MIT study every student has to pass the basic physics courses 8.01x and 8.02x. These courses are held in TEAL classrooms enhanced by the edX platform. The x at the end of the course number indicates the use of edX. In the edX system students can find the complete course material, videos of the lessons as well as administrative details. Furthermore, students have to complete online homework, such as answering multiple choice questions or filling in text boxes. Handwritten answers are also submitted through the edX system. 8.01x and 8.02x made use of 7 TEAL simulations which were accessible through the edX platform. These simulations should support students to understand various topics of the course. Evaluations in the past have shown that many students have not used them sufficiently and some even ignored them entirely. There are various reasons for this behavior. Many students mentioned their tight schedule as studying at the MIT is very time consuming. Some students simply did not understand the experiments, others where just too lazy to work intensively with them and some where completely not interested. This led to the idea of extending the edX course as well as the simulations in such a way that the homework/exam questions have to be answered by using the virtual experiments. The desired result is that students truly understand the learning material and, as a consequence, have a strong advantage when solving homework and exam problems.

3.1. TEALsim goes edX

Each term about 800 freshman MIT students attend the physics courses 8.01 and 8.02. During the courses various experiments are presented. Giving every single student the possibility to work himself with an experiment to understand the stuff is just impossible for this amount of students. That leaded to the idea of creating virtual simulations which are accessible for every student. In the first place about 20 simulations of the physics courses 8.01 and 8.02 where implemented in a stand alone software called TEALsimulation or short TEALsim which was written in Java. The goal was to give all the students

the possibility of working with physics experiments and understanding the course stuff. The drawback of TEALsim was the fact, that the students had to download the software, install an Java SDK and work on their own on the simulations with no hints and background information about what is going on in the simulation.
As the Internet got more and more normal to our live, many courses at the MIT where held with edX as well as the two basic physics courses which where then called 8.01x and 8.02x with the x indicating that they are edX courses. The next consequent step was to integrate the TEAL simulations into the edX courses. For a first test 7 simulations where ported to edX using the Google Web Toolkit (GWT) and JavaScript. Evaluations of the courses have shown that the students still hate the physics courses but the really liked the benefits of edX. Especially the fact that they had to answer online questions and submit homeworks in edX and immediately got a feedback to their answers where the best thing for them as it made learning very efficient. But as mentioned at the beginning of this chapter the evaluations also showed that the students did not really made use of the simulations. This leaded to the desire of making the simulations more interactive with edX which is the topic of this master thesis.

3.2. The goal of this master thesis

The main problem of the TEAL simulations in edX was their lacking of interactivity between them and edX. The goal of this master thesis was to create an interface between edX and the simulations for giving the course creator the possibility of asking students questions which have to be answered by using the simulations. This led to four requirements. The simulations needed to -

1. - be adapted for being able to ask useful questions.
2. - get an initial state from edX.
3. - be able to return an answer to edX.
4. - be able to reload a previous state giving the student the possibility of making a break and continue working at a later point.

The most important requirement for the interface was to keep the simulations strictly separated from the edX course. The reason behind this claim was the fact that for 8.01x and 8.02x there was a course creator and a simulation creator and the interface should provide the interaction between them. Ideally that means that the course creator has no knowledge about the simulation and the simulation creator no knowledge about the course. They should each other just once at the beginning of creating a new simulation for a course where they

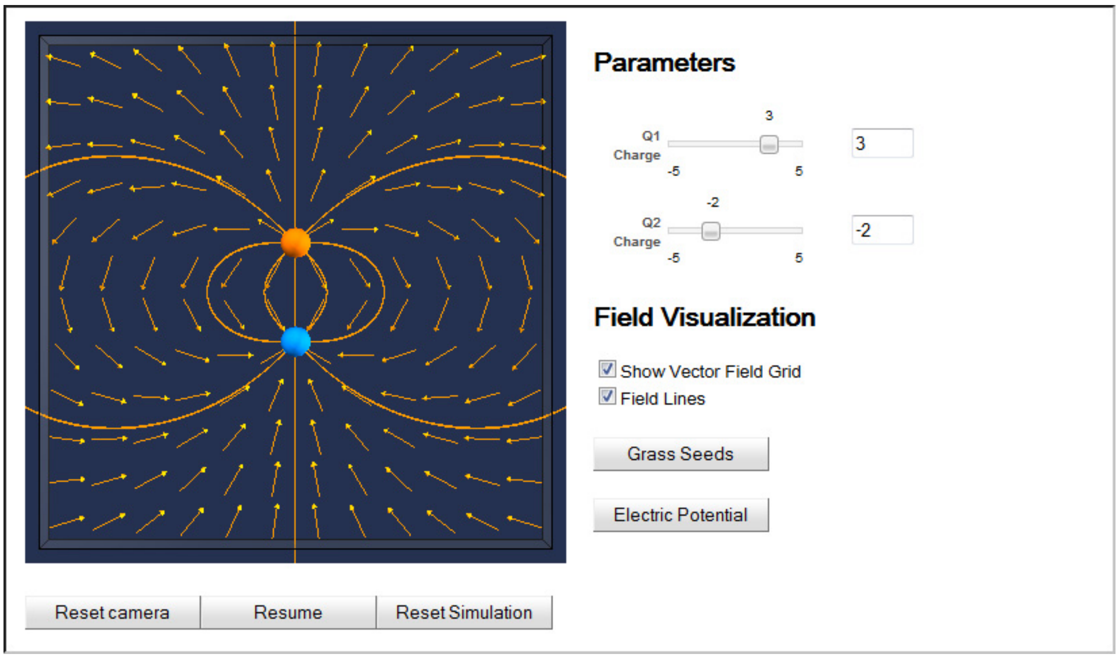


Figure 3.1.: The original point charges simulation.

discuss how the course should look like, what the simulation should do and finally which data should be sent between the course page and the simulation.

The first simulation we wanted to adapt was the point charges experiment which we will use as an example here. The experiment can be seen in figure 3.1. It has two point charges which can be placed anywhere within the dark blue box. After starting the simulation the point charges will start moving depending on the influence of their charge on each other. Furthermore the field lines are shown. Using the sliders the student can adjust the charge of the point charges. This simulation is well done for understanding the interaction between point charges but only as long as a student takes the time to work with it.

3.2.1. Adapting the simulations for answering questions

The first thing we had to do was to find a the questions we wanted to ask students and how the simulations had to be adapted to answer them. The questions should be in a way that they make the students work intensively with the simulations, understand the physically processes, be well prepared for the exam at the end of the term and ideally get a knowledge for the rest of his life.

Observing our example, the point charges experiment, we can see that there are no questions which can be answered with it. So we had to adapt it. A first idea was to hide one point charge and the student has to find it by moving the second charge. He then places a pointer where he thinks the hidden charge is located and submit his answer.

3.2.2. Getting an initial state from edX to the simulation

Taking the example of the previous chapter 3.2.1 we will quickly find the first problem namely cheating. Students will of course work and learn together. If every student gets the same question, which would mean every student get the same place of the hidden charge, it would make it to easy for them to cheat and the learning effect would be gone. Therefore every student has to get a different place of the hidden charge. EdX is here really helpful. In an edX question we are able to create random numbers moreover we can decide if we want just one random number per student per experiment no matter how often he repeats the simulation, or if we want a new random number on every attempt. We just needed to to get this random number into our simulation, that implies we need an initial state for our simulations.

3.2.3. Returning an answer to edX

Once the student gave his answer we had to grade it. This has to be done by edX for two reasons. First, as we described in the introduction of this chapter, we want to give the course creator as much freedom at creating a question set as possible. For example that gives the course creator the possibility of deciding how exact the place of the hidden charge has to be found to be marked as correct. Furthermore again edX provides us some very useful advantages. The most important is, edX saves the answers of the student. And even more, it saves all given answers as well as how much time the student has invested to find the answer. That gives the course creator an overview how hard or easy it was for all the students to find the answer and how he should adjust questioning for the next term. Therefor we needed to return the answer from our simulation to edX.

3.2.4. Reloading a previous state

It is an important feature of edX courses that a student can make a break during a question set or maybe edit his answers till the deadline and we also wanted this feature for our simulations and again edX helps us reaching this aim. EdX not only provides a *Grade* button, it also provides a *Save* button. Compared to each other this buttons do mostly the same. Both return the current state of the simulation to edX and save it. The only difference is, that at save the answer is not graded. But as said, both save the answer. So for reloading we just need to load our simulation not with the initial state but with the save state.

We will get in more detail of this points in chapter 5.

3.3. Summary

In this chapter we describe the points we wanted to achieve within this master thesis. MIT basic physics courses 8.01x and 8.02x were held with the edX platform, using seven virtual physics experiments, so called TEAL simulations, for teaching and helping students understanding the topics. These simulations are working stand alone in an embedded html page with no possibility of interacting with edX and no option of asking students questions which have to be answered by using them. The first problem was to find an as effective as possible way of adapting the existing simulations for a suitable questioning. With this knowledge we had to realize an interface between edX and the simulations with the claim to provided everything needed for a stable communication between them. Therfore the interface had to support loading an initial state, returning the students answer as well as saving and reloading the current state of the simulation for giving a student the possibility of resuming his work after a break. Another very important task was to keep the strict separation between course creator and simulation creator as these are two different persons for the physics courses who were not familiar with each other's tasks. We describe our work on the first simulation we extended during this master thesis, the point charges simulation, which can be seen in figure 3.1.

4. Related work

The main reason for this thesis was that, to the best of our knowledge when this work was written, there did not exist something similar. As described in 4.2, there was the online course STAR-Biology which would have fitted our needs but their implementation concept was not suitable for us.
J.M Claus from edX suggested the use of jsinput. Unfortunately, jsinput was available only as a beta version and did not provide enough functionality for our simulations. Additionally, as described 4.1, the synchronization with GWT was problematic.

4.1. Similar concept but different problem - The jsinput sample problem

Jsinput was created to give course creators the possibility of creating interactive JavaScript simulations and to let the students answers be graded and stored by edX. It is an interface between edX and the JavaScript simulation. Therefore, jsinput includes three functions: GetGrade, setState and getState. GetGrade can be used for returning an answer of a student from the JavaScript simulation to edX and grade it afterwards. SetState and getState can be used for saving and restoring the current state of the simulation. For more details on this functions see section 5.1.2. At the time we worked on this master thesis jsinput was still in development and there existed no online course which used it. Furthermore there was no up to date and complete documentation, only a more or less stable working sample problem, written by J.M. Claus who is responsible for the development of jsinput. Due to this we had to work only with this sample problem, write our own tests and relay on the help of J.M. Claus. Though jsinput looked quite useful for our implementation but there were two problems. First: We needed the option of setting an initial state for our simulations as described in section 3, which was not possible with jsinput. Second: Our simulations were written in GWT. This may not look like a problem because GWT compiles the Java code to JavaScript but there was a synchronization problem. That means

jsinput would try to call functions from our compiled JavaScript code which may not have been loaded at that time. The reason for that can be found in the code optimization of GWT. For a detailed description of this problem and our solution see section 5.5. For us that meant a plain jsinput implementation would not work.
A more detailed description of jsinput can be found in the edX-manual [8] and also in section 5.1.2 of this work.

4.2. Different concept but similar problem - The edX course STAR-Biology by Ivica Ceraj

It would take way too long to describe the implementation of the online course STAR-Biology. The reason is that it was created a while ago when jsinput was not yet implemented or at least only in early beta stage. STAR-Biology was mainly written in GWT so it also struggled with the synchronization problem. Hence, Ivica Ceraj decided to build the whole communication between edX and the GWT simulation on his own. The result is a huge library which provides everything for a stable interaction between GWT and edX. But there was a major difference too our work. As we describe in section 3, we needed a strict separation between the course creator and the simulation creator. Furthermore both did not want to write JavaScript code. At STAR-Biology that was different. The whole course was written by only one person which additionally had outstanding JavaScript knowledge. Thus it had no separate interface. The code for interacting between edX and the simulation is rather spread over the course xml file, the GWT simulation and a HTML page connecting them. Therefor we could not adopt this concept for our course.

4.3. Summary

For the implementation of our interactive simulations we found two different concepts as reference. First: Jsinput, which provided an interface for the interaction between edX and a JavaScript simulation. Even so, at the time we worked on this thesis jsinput was still in development and also lacking functionality we needed. Second: STAR-Biology, an edX course that also featured interactive simulations which are graded by edX. Since the source code of this course had no strict separation between the edX course and the interactive GWT simulation the concept and implementation did not fulfill our requirements .

5. Implementation of the interface between TEAL simulations and edX

Basis for our work was the course 8.02x from 2013. This course contained seven TEAL simulations which where used in so called pre-class assignments. As described in chapter 3.2 these simulations should be extended for asking students questions which had to be answered by using them.

The first simulation we wanted to extend was the *Point Charges* simulation. The original Point Charges simulation can be seen in figure 2.6.2. Initially it had two point charges, a positive and a negative one. Principally the student could move them within the bounding box and observe the field pattern created by the charges.

From the programming point of view, the edX course page including the simulation is a problem-set XML file. This file contains the whole course text for simulation description and the background and everything else. Furthermore it embeds the simulation with an iframe element which is HTML5 compliant. The structure of this page is illustrated in 5.1.

As described in chapter 2.7 the simulations are written in Java and compiled with GWT to JavaScript. Therefore GWT creates a HTML host page which then embeds the JavaScript simulation. That means, in the edX course XML page a normal HTML page is embedded no matter what this page contains or how this page is created. For edX every HTML page is same, it will be embedded statically and there is no possible interaction between them.

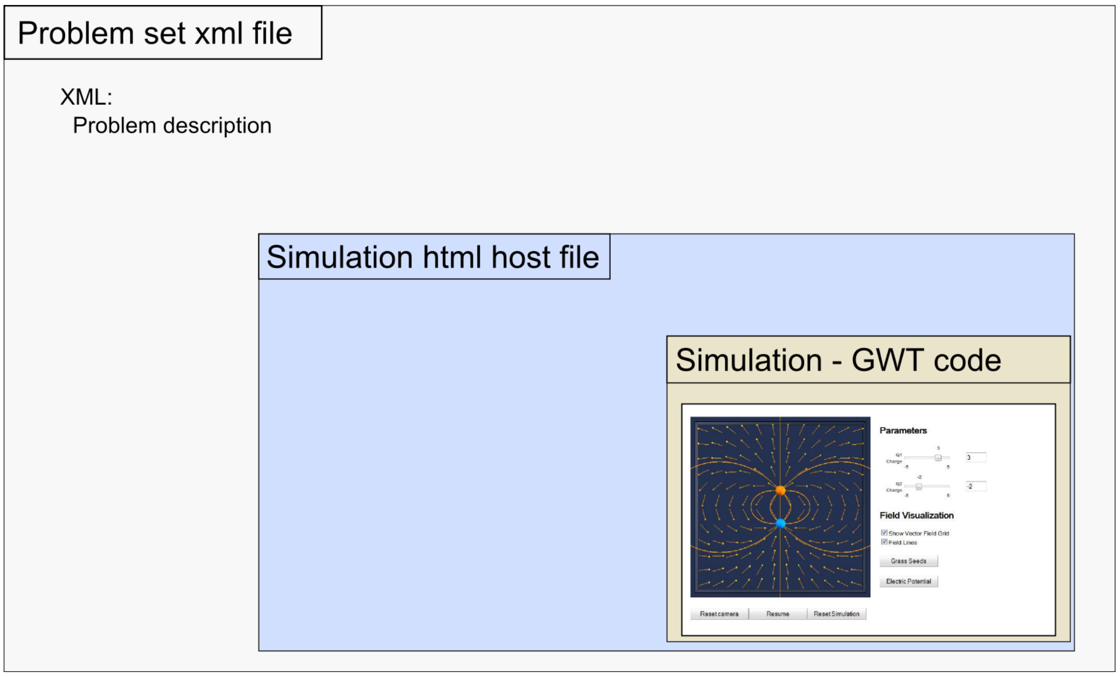


Figure 5.1.: Structure of an edX course page which includes a non interactive simulation.

The first extended simulation

For better understanding in this work we will call our modified simulations *extended simulations*. Figure 3.1 shows the basis for our fist simulation we wanted to extend, the point charges simulation. This simulation should be changed to give students an assignment which has to be solved by using it.

The idea for the new simulation was to place two hidden charges and the student has to find them by moving a visible point charge and observing the field lines. The extended simulation can be seen in figure 5.2.

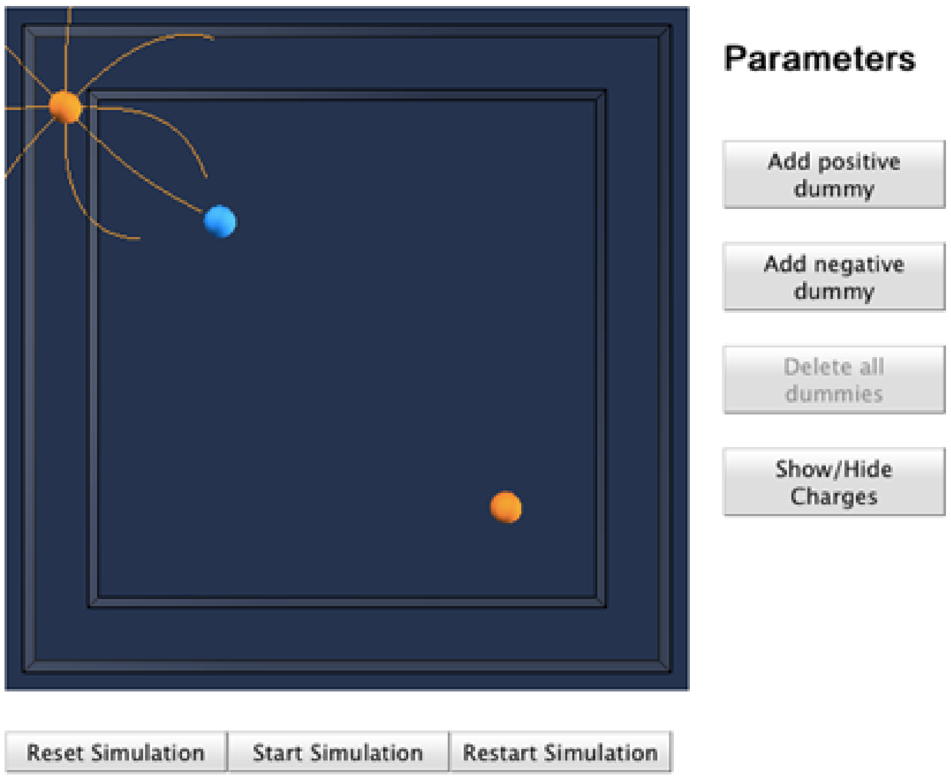


Figure 5.2.: The first extended simulation. The point charge on the left top is the movable point charge with visible field lines. The two others are dummy charges with no effect on the electrical field.

The extended simulation has two boxes. The inner box describes the area where the hidden charges will be placed. Between the inner and the outer box is the area where the moving point charge is placed and can be moved. That means, the student is not able to move the point charge inside the inner box, what would make the task too easy. Instead he has to observe carefully the field lines of the moving charge to predict the places of the two hidden charges. For answering, the student can place so called dummy charges. This are point charges with no effect on the electrical field indicating where the student assumes the hidden point charges are placed.

The requirements on the extended simulations are:

- Place hidden charges. For making cheating for students impossible the hidden point charges should be placed randomly. The random places should be created by edX and sent to the simulation on starting it. Every

student will get a unique set of hidden charges. The simulation will get the initial values for the hidden charges from edX.
- When the student presses edX' Check button, the simulation has to send the places of the dummy charges to edX which then verifies the answer by comparing the places of the hidden charges with the dummy charges.
- Enabling or disabling the Show/Hide button. This button can toggle the visibility of the hidden charges. For learning assignments this button can be enabled, for exam assignments it will be disabled.

5.1. Grading a simulation

The first task of our work was the grading of a students given answer by using the simulation. A GWT created simulation contains a HTML host page which then embeds the JavaScript simulation. J.M Claus from edX told us to use a edX command called *jsinput*. Jsinput was implemented to communicate with JavaScript applications embedded in an edX course. More precisely it embeds a HTML host page containing the JavaScript application and calls JavaScript functions within the HTML host file for communicating. This functions do not have to be part of the application as long as they exist and accept and return the correct parameters. For us that meant for the beginning we will reduce the embedded simulation to the HTML host file holding variables for manual created answers. This variables will then be updated later by our simulation but for now it is enough to have them static as they simplify the task.

For the beginning, the HTML host file has two variables:

1. *positions_hidden_charges* - This is a list of the x and y coordinate of the hidden charges as well as their charge.
2. *positions_dummy_charges* - This is a list of the x and y coordinate of the dummy charges as well as their charge.

As mentioned before we will use *jsinput* for communicating with the HTML host page. Jsinput itself has to be enclosed in a edX command called customresponse. We will describe this two functions in the next chapters 5.1.1 and 5.1.2. Applied to the example in the introduction of this chapter the structure has changed as shown in figure 5.3.

The problem XML file contains now the customresponse tag that includes the jsinput tag. Jsinput embeds then the HTML host file. Furthermore the XML file includes a python function for grading an answer. The data stream in detail: Jsinput communicates with the HTML host file. When a student clicks the *Check*

```
<problem>
  <script type="loncapa/python">
    def grade_answer(expect, answer):
    sum = int(answer[0])
    if sum==int(expect):
      return{'ok': True, 'msg': 'Right␣answer.'}
    else:
      return{'ok': False, 'msg': 'Wrong␣answer.'}
  </script>
  <p>What is the sum of 4 + 6 ?</p>
  <customresponse cfn="grade_answer" expect="10">
    <textline size="10/>
␣␣</customresponse>
</problem>
```

Listing 5.1: A Customresponse sample problem, based on [8].

button in the edX page jsinput will call the function *grade_function* of the HTML host page which will return the value of the variable *position_dummy_charges* which holds the students answer. Customresponse calls the python grade function of the edX XML page which grades it. The output of this python grade function will be processed by edX. That means for example edX saves the answer, it records all given answers for evaluations or give the student a mark based on the answer. All this is done by edX itself, the python grade function has just to determine if the given answer is right or wrong.

5.1.1. EdX - customresponse

Customresponse is used to evaluate student answers using a python script. A code example can be seen in 5.1. The customresponse tag contains a single text-line where a student can enter his answer. The exercise in this example is to calculate the sum of 6 plus 4 and enter the answer. For grading, customresponse takes two parameters. The first *cfn* takes the name of the python check-function. The second *expect* is the expected answer. *Expect* does not have to be set. If not, the check function has to get the expected from somewhere else or calculate it on its own.

In this example, when a student enters an answer and presses the check button customresponse will take the entered value and call the check function with it. The python function then compares the given answer with the expected answer and returns the either *Right answer* or *Wrong answer*.

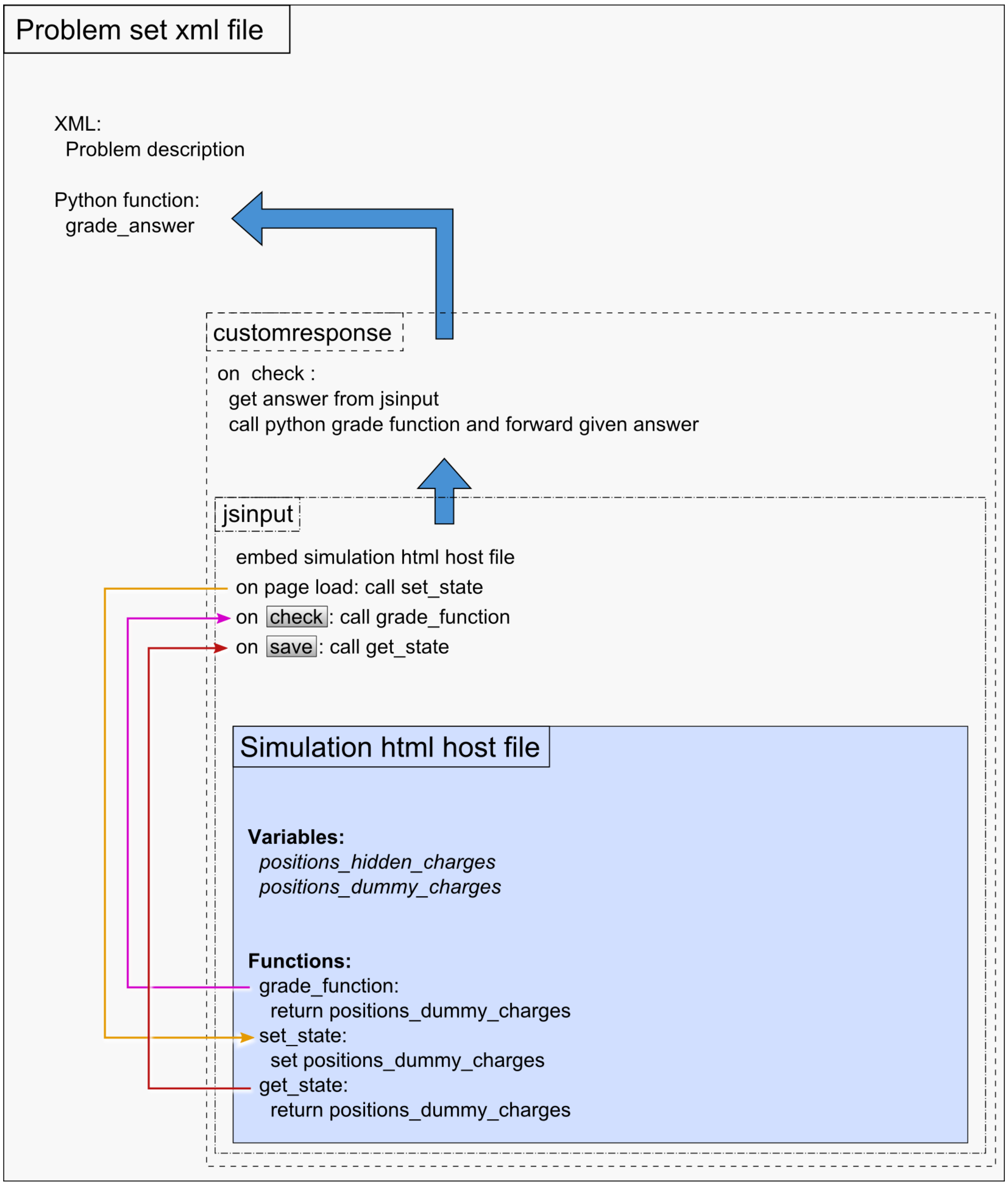

Problem set xml file
XML:
Problem description
Python function:
grade_answer
customresponse
on check :
get answer from jsinput
call python grade function and forward given answer
jsinput
embed simulation html host file
on page load: call set_state
on check: call grade_function
on save: call get_state
Simulation html host file
Variables:
positions_hidden_charges
positions_dummy_charges
Functions:
grade_function:
return positions_dummy_charges
set_state:
set positions_dummy_charges
get_state:
return positions_dummy_charges

Figure 5.3.: Structure of a course page that embeds a HTML file using jsinput. Illustrating the data stream.

5.1.2. EdX - jsinput

Jsinput is a method for getting edX communicate with standalone HTML files. This is useful if an application is written in JavaScript and should be embedded in an edX problem. Therefore any application like GWT can be used for creating the simulation. Jsinput provides three functions for interacting with a HTML file. They can be used for grading students work with the simulation, saving and reloading it. The three functions in detail:

- gradefn: for grading students work. This is a JavaScript function that returns the students answer. It is not grading it!
- get_state: for getting the current state of students work for saving it. It returns the current state of the simulation. The difference to *gradefn* is, that the state can be more complicated than the students answer and therefore returning more values can be required. For example, if we would ask in our extended simulation just for the overall charge, the answer would be just a number. But for reloading the simulation after a brake also the positions of the dummy charges are needed. In this case gradefn would return the number and get_state the positions of the dummy charges.
- set_state: for reloading students previous work. This function takes input values for reloading a previous state.

This leads to a clear separation of edX and the JavaScript simulation and is perfect for our case where the edX course and the physics simulations are written by different people. It means for the developer of the simulation that he has just do implement these three functions and does not need to worry about grading, evaluating, student's identification, state saving and so on. This is all done by edX. On the other hand for the developer of the edX course it means he can be sure that there exist these three functions in the HTML file. He can use them without knowing how the simulation is created, how the students answer is prepared or anything else. He gets the answer and can evaluate it using known edX XML and python code.

An example code for a minimal jsinput problem set can be seen in 5.2.

The parameters jsinput can take are the three functions as described above as well as:

- height and width: two integer values that set the size of the area the included HTML page will get in the edX course
- html_file: the source of the embedded HTML file

The parameters and their properties can be seen in table 5.1.

```
<?xml version="1.0"?>
<problem>
  <script type="loncapa/python">
    import json
    def grade_answer(expect, ans):
      answer_and_state = json.loads(ans)
      answer = json.loads(answer_and_state["answer"])
      x = int(answer['x'])
      if x == expect:
        return{'ok': True, 'msg': 'Good Job!'}
      else:
        return{'ok': False, 'msg': 'Try again.'}
  </script>
  <customresponse cfn="grade_answer" expect="3">
    <jsinput
      gradefn="gradefn"
      set_statefn="setstate"
      get_statefn="getstate"
      width="818"
      height="700"
      html_file="/static/simulation.html?size=400"/>
  </customresponse>
</problem>
```

Listing 5.2: A jsinput sample problem, based on [8].

Attribute Name	Value Type	Required	Default
html_file	Url string	Yes	None
gradefn	Function name	Yes	gradefn
set_statefn	Function name	Yes	None
get_statefn	Function name	No	None
height	Integer	No	500
width	Integer	No	400

Table 5.1.: Attributes for jsinput [8].

5.1.3. Grading using python

In the previous sections we described how we get the given answer from the HTML host file to the edX XML course. For evaluating the answer python is used. The code example is the same as for jsinput 5.2. The defined grade function is called *grade_answer* and takes the two parameters *expect* ant *ans*.

The structure of the parameter *ans* is depending on the use of jsinput. **If *get_state* and *set_state* are not set**, *ans* contains only the given answer as a string. **If *get_state* and *set_state* are set** the answer is a json string containing the answer **and** state!

For our example code 5.2 that means, in line 6 the value of the parameter ans is loaded in *answer_and_state* which contains now as the name says the state **and** the answer. As we are just interested in the answer string we store it in a separate variable called *answer* as seen in line 7. This variable now holds the given answer as a list of values. In line 8 we take a specific value called *x* which will then be compared to the expected variable. Finally we return *True* or *False* depending on the given answer.

5.2. Starting a simulation with initial values

The previous section 5.1 describes how to get data from the HTML host file to the edX XML file. In this section the other way around is explained. The requirement was to create random variables for the positions of the hidden charges in the edX XML file and send them to the HTML host file. The reason why the values have to be created in the edX XML file is that edX offers some advantages which should be utilized. One is the capability of setting if edX should create a new random variable on each attempt a student makes or just once for every student and then this variable will on every attempt be the same. This is important for our work as we wanted just one randomly created set of hidden charges positions per student. This give the student the possibility of checking if his answer is correct and on a fail retry to find them. The other reason was to give the course creator the possibility of having influence on the positions of the hidden charges and hence on the difficulty of the assignment. For our extended simulation that means the course creator can limit the range of the randomly created positions. This is important as hidden charges near to the moving charge are easier to find then centered charges.

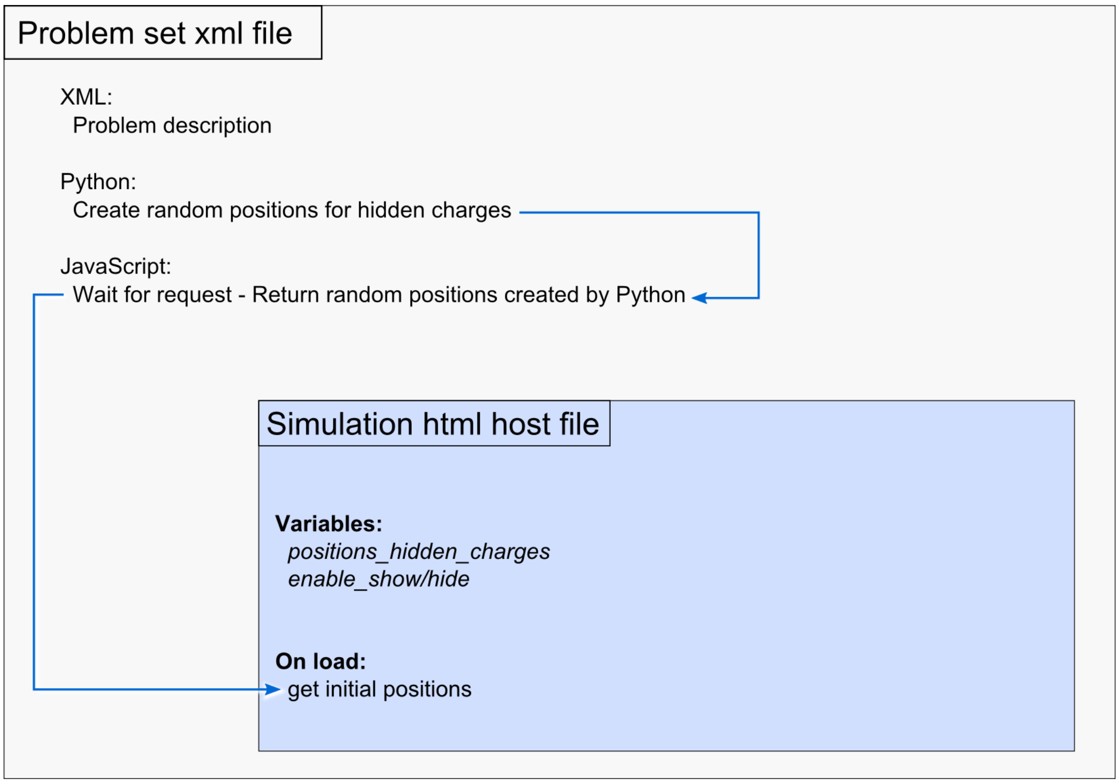


Figure 5.4.: Structure of a course page that embeds a HTML file using postmessages for sending the values of the initial positions of the hidden charges to the HTML host file.

Again we extend our initial example 5.1 with the new methods. The resulting structure and data stream can be found in figure 5.4. As in our example for grading the students answer we work only with the simulations HTML host file, hiding the simulation itself for better understanding and as it is not involved in the communication.

5.2.1. Creating the initial values with python

The positions of the hidden charges are created by python function in the edX course XML file. In the final simulations this positions should be randomly created. For our prototype simulation we created static positions as it would make testing easier. An python code example for randomly created values are shown in 5.3. Note the first line. Here we tell edX to create random variable just once for every student. The example code creates the positions and charges for two point charges, where the possible position is between -5 and 5 and the charge is either -1 or 1.

```

<?xml version="1.0"?>
<problem display_name="Point Charges Extended" rerandomize="
    per_student" attempts="1000">

  <script type="loncapa/python">
    hidden_charges = numpy.zeros((2,3))

    hidden_charges[0][0] = random.randint(-5,5)
    hidden_charges[0][1] = random.randint(-5,5)
    hidden_charges[0][2] = random.choice[-1,1]

    hidden_charges[1][0] = random.randint(-5,5)
    hidden_charges[1][1] = random.randint(-5,5)
    hidden_charges[1][2] = random.choice[-1,1]
  </script>

...
```

Listing 5.3: Creating randomly positions for two hidden charges using python.

5.2.2. Iframes and postmessages

For getting the created positions from the edX course XML file to the simulations HTML host file the JavaScript postmessage method is used. By using the jsinput function, the HTML host file will be embedded with an iframe HTML tag. For security reasons a direct access to JavaScript variables in the host file is not allowed as the embedded site can also be on a unknown server. Hence messages are sent for communication. A benefit of this circumstance is that our extended simulations could be stored on an external server also.

The postmessage method always needs a sender and a receiver. The receiver listens for a message which causes a so called message event.

The communication between our edX XML file and our HTML host file is as followed. Important is the exact chronology!

1. The edX XML page is called
2. The initial values for the positions of the hidden charges are created by the XML file
3. An eventlistener is started by the XML file
4. The HTML host page is embedded by the XML file
5. The HTML host page also starts an eventlistener

```
1
2  <script type="text/javascript">
3    initRespond = function(e)
4    {
5      e.source.postMessage( $hidden_charges[0][0] + ':' + $
          hidden_charges[0][1] +  ... , e.origin );
6    }
7
8    window.addEventListener('message', initRespond);
9  </script>
```

Listing 5.4: JavaScript code for postmessage communication of the edX XML file

6. The HTML host file sends a message to the XML file requesting the initial values
7. The eventlistener of the XML page receivs the message and calls the message-event function.
8. The message-event function of the XML file sends a message with the initial values back to the HTML host file.

The JavaScript code of the edX xml file can be seen in 5.4. First the message event function is created and below in line 8 the event listener is implemented. The syntax of the eventlistener:

element.addEventListener(event_type, message_event);

The line begins with *element* which defines the element where the listener is added to, in our case the current window. *Event_type* a string describing is the type of the sent event which could be a click event for buttons or as in our case a message. *message_event* is the function that will be called when a message is received, for our example the defined *initResponse* function.

The *initResponse* function is also just a message sender which gets a single parameter *e* that is used for sending a message back to the sender of the initial message. The message itself is a single string containing all values of the initial positions separated by a colon.

The JavaScript code of the HTML host file is very same to the XML file and can be seen in 5.5.

The sender in line 14 just sends a message to everyone who listens, which is defined by "*".

```

handleResponse = function(e)
{
  positions_hidden_charges[0][0] = e.data.split(':')[0];
  positions_hidden_charges[0][1] = e.data.split(':')[1];
  positions_hidden_charges[0][2] = e.data.split(':')[2];

  positions_hidden_charges[1][0] = e.data.split(':')[3];
  positions_hidden_charges[1][1] = e.data.split(':')[4];
  positions_hidden_charges[1][2] = e.data.split(':')[5];
}

window.addEventListener('message', handleResponse);
parent.postMessage('message', "*");
```

Listing 5.5: JavaScript code for postmessage communication of the HTML host file

The message-event is again a simple function which takes one parameter including the received message. As described above it is a string containing all values for the positions of the hidden charges. The function takes this string, splits it at every colon and stores the values in the local variable *positions_hidden_charges*.

5.3. Implementation of a test course using jsinput

The previous sections 5.1 and 5.2 describe the communication between the edX XML file and the HTML host file of the simulation. Each section describes one direction of the communication. In this section we want to merge both of them and finally embed the actual simulation. Therefore we extend our structure example again to the final test course which is shown in 5.5.

The shown structure consists of the previous described communication between the edX XML file and the simulations HTML host file. Additionally the simulation which is written in Java is embedded and its communication with the HTML host file is described.

When a student clicks the simulations button *Start Simulation*, the simulation loads the initial values for the positions for the hidden charges and place them. Furthermore it enables or disables the *Show/Hide* Button depending on the value of *enable_show/hide*.

When a student clicks the *Restart Simulation* button, the same procedure as with *Start Simulation* is processed. Additionally dummy charges are placed

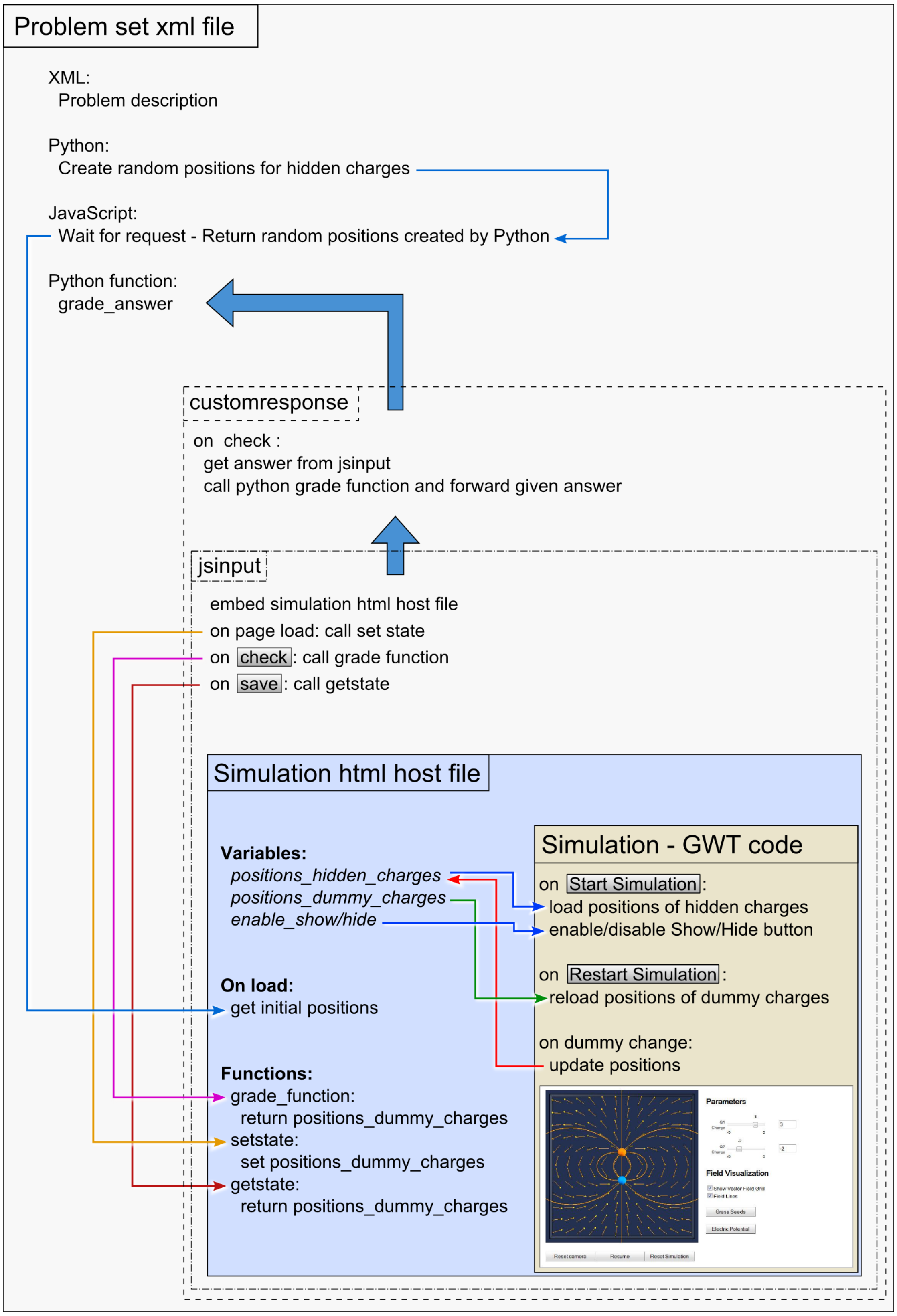


Figure 5.5.: Structure of the final test course page that embeds a HTML file.

depending on the values of *positions_dummy_charges*.

The last job our simulation has to do is to update the *positions_dummy_charges* on every placement or move of a dummy charge by the student. That ensures that at every moment during the running of the simulation the current positions of all dummy charges are saved in the HTML host file and hence jsinput can read them for grading the simulation whenever the *Check* button is clicked.

5.4. Loading, saving and sending values between the different programming languages and different files

In our work values of variables are used within different programming languages and different files. At this point we want to get more into this topic and describe the different situations we were confronted with.

5.4.1. One variable in one file used by two programming languages

As described in section 5.2 for the initial positions of our hidden charges we created random variables within the edX course XML file using python. For starting the simulation we had to send the values of this variables with JavaScript to the HTML host file. The question was, how does JavaScript get the value of the Python variable? As long as this happens within one XML file the solution is very easy. JavaScript can directly access python variables within the same file. The syntax of the code can be seen in the code example 5.6.

In this example we create the variable *python_variable* using python. Then we are printing it to the browsers console using JavaScript. Therefore we can access the Python variable by using its name with a leading "$". That is also working for saving values with JavaScript in a Python variable.

```
<problem display_name="Point Charges Extended">
  <script type="loncapa/python">
    python_variable = 4
  </script>

  <script type="text/javascript">
    console.info('The value of the python variable is ',
      $python_variable);
  </script>
</problem>
```

Listing 5.6: Example for using a Python variable with JavaScript

```
public native String getValue(int i, int j)
/*-{
  return $wnd.e_initValues[i][j];
}-*/;
```

Listing 5.7: Example for using a Python variable with JavaScript

5.4.2. One variable in two files used by the same programming language

We discussed this problem detailed in section 5.2 so we mention it here just for completion. To get the initial values from the XML course file to the HTML host file we could not directly access variables of the other file neither they were displayed via iframe in the same browser window. That is a security mechanism of JavaScript and hence very important. The solution was using postmessages for sending the values between the two files. For further information please see the mentioned section.

5.4.3. One variable in two files used by two programming languages

As we stored the values for the hidden charges directly in HTML we had to get them into the GWT code. Therefore GWT offers the *JavaScript Native Interface* (JSNI). It gives programmers the possibility of writing JavaScript code within the uncompiled Java Code. An example for reading a JavaScript variable within Java can be seen in 5.7.

An example for writing JavaScript variables within can be seen in 5.8.

```
public native void updateDummyPos(int id, int x)
/*-{
  $wnd.DummyPos[id] = x;
}-*/;
```

Listing 5.8: Example for using a Python variable with JavaScript

```
public static native void log( String msg )
/*-{
  console.log( msg );
}-*/;
```

Listing 5.9: Example for using a Python variable with JavaScript

At this point we want to introduce a very useful function we used for writing text in the browser's JavaScript console. The code can be seen in 5.9.

There are a few rules every JSNI method has to fulfill [3]:

- The function has to be declared as native.
- The function has to have an empty body. Furthermore it has to end with a semicolon.
- The JavaScript code has to be written within /*-{ and }-*/ .

For examples see the code examples prior in this section.

Reloading a previous state - The way from XML to JavaScript to Java

As we described, jsinput offers a the set_state function for reloading the previous stored state of the simulation. But for our special case where the simulations are written in Java there is a fact that needs further attention: The formating of strings by different programming languages.

In many tests of our interface we had problems that we where sure our code is right but communication did not work. As we mentioned debugging is a little complicated in our case so it took us quite long to figure out what went wrong. The problem where that jsinput is returning and forwarding the answers and states always as a long string.

For example if we want to send the position of this two point charges and their charge:

- ChargeId 1; X: 5; Y: 2; Charge: 1

- ChargeId 2; X:-3; Y: 3; Charge: -1

This values as a string could look like:

1,5,2,1,2,-3,3,-1

Unfortunately they can also look like this:

1,5,2,1,2,-␣3,3,-␣1

In the last string, there is a space between every "-" and the number!

Furthermore in different programming languages a string ends with a different indicators. The most common is the "\n".

It can be a problem sending strings or variables through different programming languages. Hence we want to show at our example how we did saving and reloading the simulations state. We show the data stream in figure 5.6. As seen in the figure, edX demands from *get_state* one single string. And it delivers *set_state* the same single string back. That means, our HTML host file has also send and save the values as string.

We had to decide how to handle this strings and where we want to convert them from integers and to integers. Both time we do it using JavaScript, once direct in JavaScript and once using JSNI in our Java Code.
To prevent problems with reloading and overriding the dummy positions we decided to store them twice in the HTML host file. First for the reloaded positions from a previous state. This positions are stored through the whole simulation till the next time the student stores the new state. So the student can reload the previous state as often as he wants. Second we store the current dummy positions and and update them on every move by the student.

String to Integer

The first case is the converting the string of all values to integers. This has to be done on reloading a previous state of the simulation. The edX course XML file sends this string using jsinput to our HTML host file. We store this string as it is in the HTML host file. On loading the dummy positions we split the string into substrings where every substring contains one value. The code for this can be seen in 5.10.

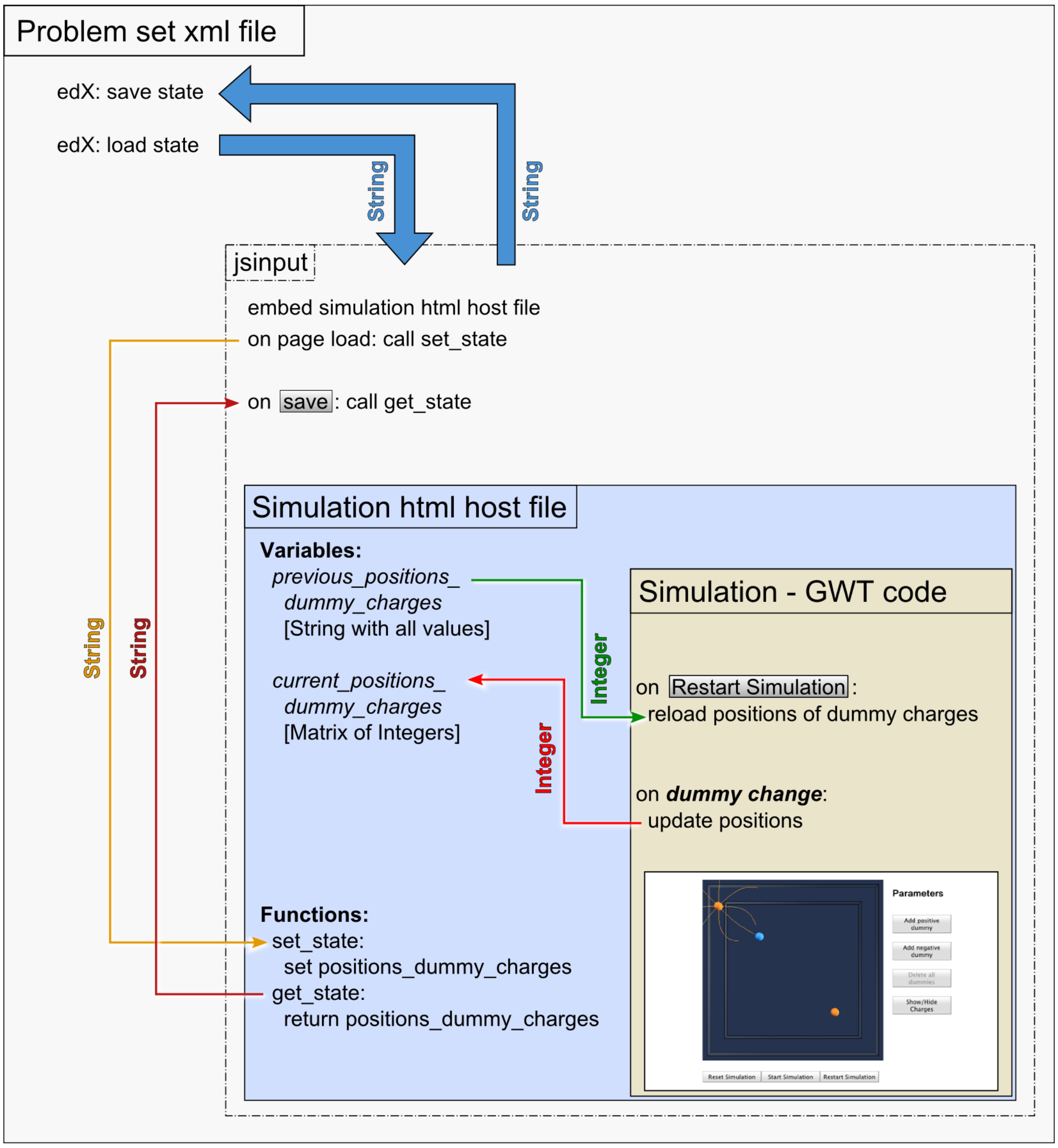


Figure 5.6.: Data stream of saving and reloading the simulations state and the used data types.

```
1 public native int getDummyValue(int i)
2 /*-{
3   var myString = $wnd.statestr;
4   myString = myString.replace(/[[\]]/g,'');
5   var myValue = myString.split(',')[i];
6   var myInt = parseInt(myValue);
7   return myInt;
8 }-*/;
```

Listing 5.10: Example for using a Python variable with JavaScript

Basically the string stored in the HTML host file contains the values separated by ",". But as we described above there can be other characters like spaces or something else. So before converting we need to clean up the string. This is done in line 4 in our code where we delete all spaces and characters which should not be in there. After that we split the string in substrings and take the i-th element.

We do that in our Java Code using JSNI for writing the code in JavaScript. The function in 5.10 is a Java function using JavaScript code as we described earlier in this section. In this function we load the string from the HTML host file. We clean it up, split it into substrings and take the i-th element. This element is converted into a integer and returned to Java.

Integer to String

The second case is to convert the integers to a single string. We have to do that twice. One in *set_state* and once in *gradefn*. Both return a single string and in particular they send the same string.

In our HTML host file we store a matrix holding the dummy positions and charge. How to save a value in the HTML host file using JNSI in Java is described in the example code 5.8. When the students presses the *Check* or *Save* button the HTML host file has to deliver a single string containing all the dummy values. Therefore we take all the values and "stringify" them with JavaScript. This can be done in two different ways. We used both of them, one for *gradefn* and one for *set_state*. The reason we did that is that the used method in *gradefn* is a little more work but better to debug. Once we were sure it is working correctly we used the simpler method in *get_state*.

The code of the function that is returning the string for *gradefn* is seen in 5.11. In line 3 we store all values in a so called *linked list*. At this point we do not

```
1  function getGrade()
2  {
3    givenAnswer = {d1x: e_curDummyPos[0][0],
4      d1y: e_curDummyPos[0][1],
5      d1c: e_curDummyPos[0][2],
6
7      d2x: e_curDummyPos[1][0],
8      d2y: e_curDummyPos[1][1],
9      d2c: e_curDummyPos[1][2],
10
11     ...
12
13     d10x: e_curDummyPos[9][0],
14     d10y: e_curDummyPos[9][1],
15     d10c: e_curDummyPos[9][2]};
16
17   console.info("getGrade␣called.␣Given␣answer:␣", JSON.stringify(
        givenAnswer));
18   return JSON.stringify(givenAnswer);
19 }
```

Listing 5.11: Returning one single string containing all positions of the dummy charges - using a linked list

want to describe linked lists as this would take too much space. Short sayed we store for every value a name. For example

d1y: e_curDummyPos[0][1]

the element *d1y* holds a value.

In line 17 we print the string to the browsers console for debugging. And in line 18 we *stringify* the linked list and return this string.

For comparison we want to show the easier code which was used for the function that is returning the string for *get_state*. It is shown in 5.12. It looks very similar to the method we used for *gradefn*. The difference is that we used an array instead of a linked list. In this case only the values are stored. This leads to less code and also less data that is sent to edX. The disadvantage is that it is less debugable data.

```
function getState() {
  givenAnswer = [e_curDummyPos[0][0],
    e_curDummyPos[0][1],
    e_curDummyPos[0][2],

    e_curDummyPos[1][0],
    e_curDummyPos[1][1],
    e_curDummyPos[1][2],

    e_curDummyPos[2][0],
    e_curDummyPos[2][1],
    e_curDummyPos[2][2],

    ...

    e_curDummyPos[9][0],
    e_curDummyPos[9][1],
    e_curDummyPos[9][2]];
  console.info("GetState called. Given answer: ", JSON.stringify(
      givenAnswer));
  return JSON.stringify(givenAnswer);
}
```

Listing 5.12: Returning one single string containing all positions of the dummy charges - using an array

For our first method using a linked list an example string would look like (shortened):

{"d1x":10,"d1y":-2,"d1c":1, ... "d10x":-10,"d10y":2,"d10c":-1}

The same example string for the second method using an array looks like (shortened):

[10,-2,1, ... ,-10,2,-1]

For getting a value out of the strings methods are different. In the case of a linked list we can select an element by it's name. In the case of an array we have to use the number of x-th element.
For example, the python code for getting the x value of the first dummy of the stringified linked list would look like:

```
dummy_1_x = int(answer['d1x'])
```

Getting the same element out of the stringified array would look like:

```
parted_answer = answer.split(',')
dummy_1_x = int(parted_answer[0])
```

5.5. The synchronization problem

At this point we want to get into an important questions which came up during our work. Why are the three JavaScript functions for jsinput implemented in the HTML host file and not directly within the GWT Java code as this code will be compiled to JavaScript code? The short answer: To ensure a stable communication. A more detailed answer is given in the next subsections 5.5.

We had different approaches for the communication during our work which where less difficult but we always found specific situations where they did not work correctly. We called that problem *The Synchronization Problem*. The reason could be found in our complex problem definition. Firstly the circumstance that we were not just embedding a plane JavaScript simulation. Our simulations are written in Java and just compiled to JavaScript. What sounds negligible emphasized as a real problem as we will describe in section 5.5. Secondly we had to keep the strict separation of the course and the simulation code.

Why are the three JavaScript functions for jsinput implemented directly in the HTML host file and not within the GWT Java code as this code will be compiled to JavaScript code?

GWT offers a feature called *JavaScript Native Interface* (JSNI). This offers the capability of writing JavaScript code directly in the Java code file. It was implemented to use JavaScript functions which are not ported or portable to Java, such as the window manipulation functions included with JavaScript. For us this feature seemed to be a real benefit, as we could implement this functions directly in our Java Code and hence they would be part of our extended simulations. That would make a direct communication between edX and our extended simulation possible without the intermediate step in the HTML host file.

We did try this approach and it seemed to work. But while testing our extended simulations we found situations where communication did not work and we could not find a comprehensible reason.

With the help of I. Ceraj we could find the problem. It could be found in the optimization of GWT while compiling the Java code to JavaScript. As every bit sent over the Internet slows an application the GWT compiler tries to keep the code as small as possible. That means on the one hand that functions which are not used could completely excluded. On the other hand a function that is not needed at starting an application could be loaded later to keep the start procedure as fast as possible. And this led to the malfunction of the communication because GWT could not know that our jsinput functions have to be present from the beginning of the extended simulation, especially the set_state function.

At the time we wrote this thesis, for our knowledge, there was no possibility to tell GWT when a function has to be ready loaded. One solution would be to implement a query if the functions are ready loaded and if not to wait for a while and try again. But this could on the one hand lead to a longer and noticeable delay for the user. On the other hand it is not good looking programming style.

The solution was to use the HTML host file as a interim stage between the edX course and the GWT simulation where the necessary values are always present and usable for both of them.

5.6. Summary

In this section we described the communication between the edX XML course file and our extended simulation created with GWT. We started with our basis, a edX course page including the point charges simulation but with no interaction between them. We then showed step by step how this basis page was extended till the final interface.

At this point we want to give another overview of the interface but with a focus on the chronology as this is very important for guarantee a stable communication. Therefore we created a flowchart which is shown in figure 5.7. It illustrates which function and which variable has to be available and updated at which point of time.

The reason why it is so important to strictly keep this chronology is that there are very few possibilities included in JavaScript for querying if a function or variable exists at the moment and furthermore no methods for dealing with this fact.

During our work we worked with programmers of edX and we were told that at the time we wrote this master thesis, jsinput was still in beta stage. Moreover we heard that jsinput is constantly improved and enhanced and for the future there will be better and easier solutions for our interface.

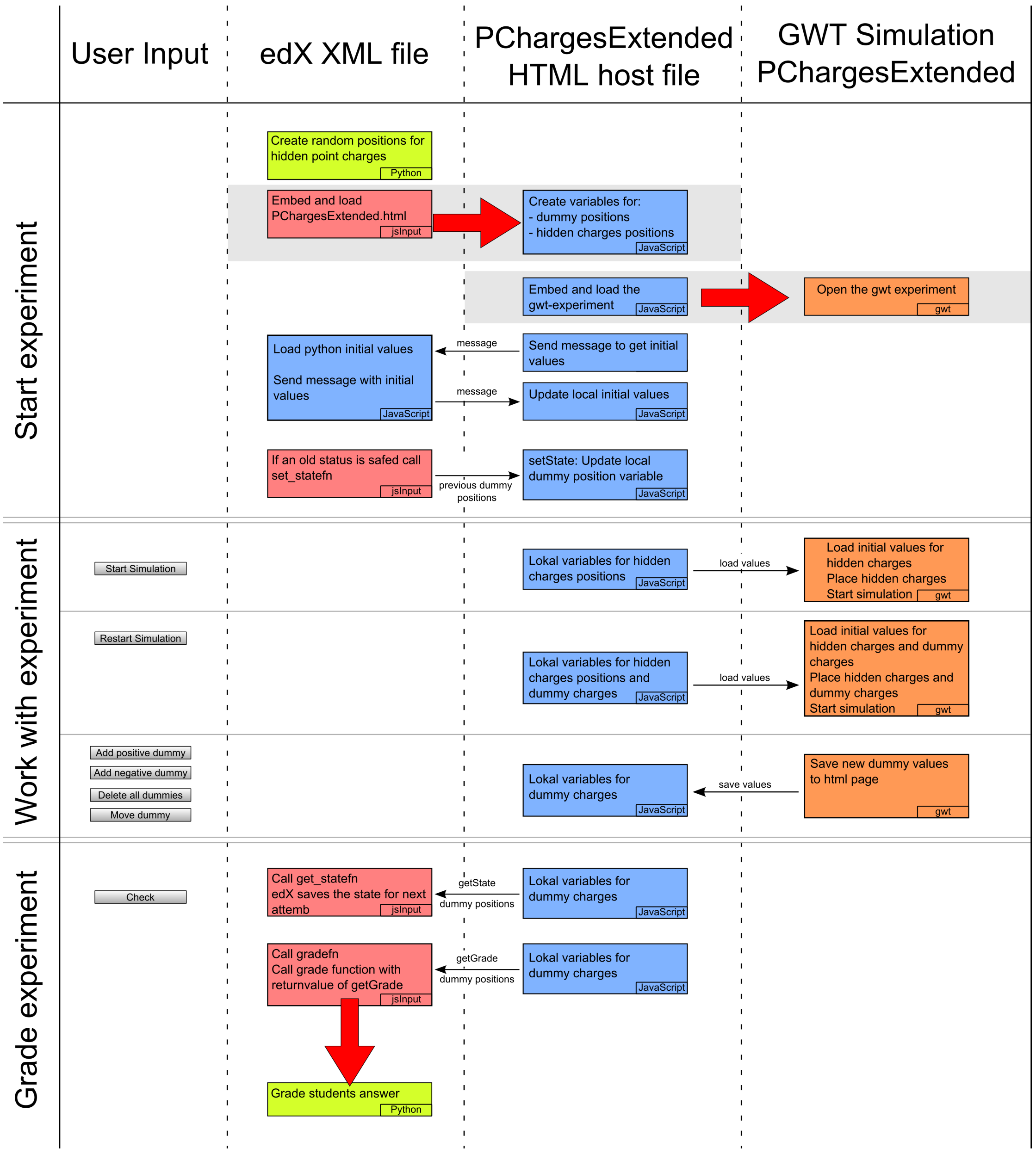

User Input
edX XML file
PChargesExtended HTML host file
GWT Simulation PChargesExtended
Start experiment
Create random positions for hidden point charges
Python
Embed and load PChargesExtended.html
jsInput
Create variables for: - dummy positions - hidden charges positions
JavaScript
Embed and load the gwt-experiment
JavaScript
Open the gwt experiment
gwt
Load python initial values
Send message with initial values
JavaScript
message
Send message to get initial values
message
Update local initial values
JavaScript
If an old status is safed call set_statefn
jsInput
previous dummy positions
setState: Update local dummy position variable
JavaScript
Work with experiment
Start Simulation
Lokal variables for hidden charges positions
JavaScript
load values
Load initial values for hidden charges
Place hidden charges
Start simulation
gwt
Restart Simulation
Lokal variables for hidden charges positions and dummy charges
JavaScript
load values
Load initial values for hidden charges and dummy charges
Place hidden charges and dummy charges
Start simulation
gwt
Add positive dummy
Add negative dummy
Delete all dummies
Move dummy
Lokal variables for dummy charges
JavaScript
save values
Save new dummy values to html page
gwt
Grade experiment
Check
Call get_statefn
edX saves the state for next attemb
jsInput
getState
dummy positions
Lokal variables for dummy charges
JavaScript
Call gradefn
Call grade function with returnvalue of getGrade
jsInput
getGrade
dummy positions
Lokal variables for dummy charges
JavaScript
Grade students answer
Python

Figure 5.7.: Flowchart of the complete communication between edX and our extended simulation.

6. The converted TEAL simulations

The goal for this master thesis was to create an interface between edX and the simulations and furthermore to extend one TEAL simulation for answering questions using it. Luckily we had enough time to extend two more simulations. The three simulations in chronological order:

- Point Charges Extended - Section 6.1
- Gausses Law Extended - Section 6.2
- Amperes Law Extended - Section 6.3

The order was based on the order of their occurrence in the edX course 8.02x.

After we had created a working example simulation as described in the chapters before we had to think about how to extend an existing simulation to ask educationally meaningful questions which have to be answered by using the new extended simulation. We realized that this has to be done for every simulation on its own as they are too different.

What all extended simulations should have in common is the fact that they should be as simple or, as we called it "*as stupid*", as possible. That means, we wanted to give the course creator as many possibilities and latitude as possible that he can use them in different ways for different questions and with different difficulties. One idea was to use various problem-sets to explain the physical laws behind the simulation where every set could be used to describe specific effects for different situations. Another idea was to increase difficulty with every problem set to slowly explain problems.

6.1. The first extended TEAL simulation: Point Charges Extended

We already extended this simulation for the prototype and the tests as described in the previous chapters. Hence at this point we had a principally working simulation which was able to communicate with an edX course. Now we had to define a real problem set with a meaningful simulation.

6.1.1. Requirements

When the student starts a simulation a number of hidden charges should be placed in the inner box. A visible positive charge is placed between the inner and outer box which can be moved around the inner box. This charge has visible field-lines with finite length. By moving the visible charge the student should find the hidden charges. Therefore he can place dummy charges with no effect on the field-lines to mark the places of the hidden charges. When the student checks his answer the positions of the placed dummies is sent to the XML course file where the answer is graded.

The number of hidden charges should be determined by the course creator within the XML course code using python. The maximum of hidden charges should be five as we thought this would be the highest number of charges that fits in our simulation space.

The places should also be determined by the course creator within the XML course file so he can decide how difficult the charges should be placed. Again python should be used as this would give the course creator the possibility of random created places. With python edX can set every random number to *one per student* or *one per attempt* as described in chapter 5.2.

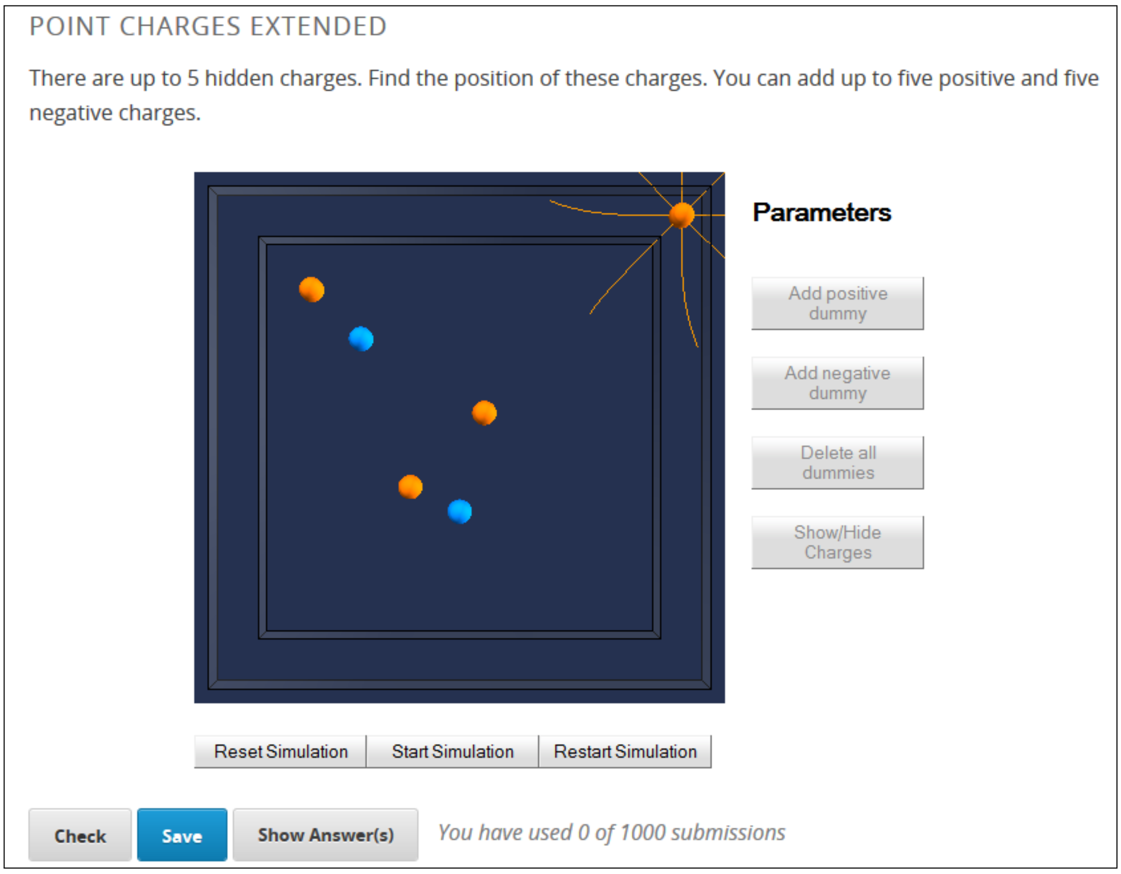
The dummies should be only placeable on a grid as otherwise it would be too hard to find the hidden charges. The size of the grid has to be adjusted while testings.

The grading should be done by python in the XML file to utilize all benefits of edX like statistics, overall course marks and others as described in chapter 5.1.

A Show/Hide Button should be implemented which shows and hides the hidden charges. The course creator should get the possibility of turning this button on or of for every problem set so he can create learning sets including it and exam sets excluding it.

6.1.2. The simulation

Figure 6.1 shows the final *Point Charges Extended* - Simulation. It displays the started simulation with the moveable charge on the right top. Within the inner box the five *hidden charges* are shown what can be done by the Show/Hide button next to the simulation on the right.


POINT CHARGES EXTENDED

There are up to 5 hidden charges. Find the position of these charges. You can add up to five positive and five negative charges.

You have used 0 of 1000 submissions

Figure 6.1.: The final *Point Charges Extended* - Simulation

The simulation in figure 6.1 shows our test set for the hidden charges. The idea was to place the maximum number of hidden charges with different difficult places. Therefore we placed charges near the border and near the center as well as near to each other and more separate.

6.1.3. The buttons in detail

- **Start Simulation:** Starts the simulation. Loads and places the hidden charges and enables the 4 dummy buttons.
- **Restart Simulation:** Loads and places the hidden charges, reloads previous set dummy charges and enables the 4 dummy buttons.
- **Reset Simulation:** Resets the view of the simulation as it can be rotated.
- **Add positive dummy:** Places a positive dummy which can be used for marking a hidden charge. This button is only enabled till 5 positive dummy charges.
- **Add negative dummy:** Places a negative dummy which can be used for marking a hidden charge. This button is only enabled till 5 negative dummy charges.
- **Delete all dummies:** Deletes all dummies from the simulation. This button is disabled if no dummies are placed.
- **Show/Hide Charges:** It shows and hides the hidden charges. This button can be disabled by the course creator for exam problem-sets.
- **Check:** With this button the students answer is checked and graded. The places of the dummy charges are sent to the edX XML course file and grade there by python. Afterwards a notification shows the student how many charges he found correctly. Furthermore the students answer is stored and can be reloaded whenever he wants. This button is only enabled till the *Due Date* of the problem set.
- **Save:** Saves the students answer. It saves the positions of the hidden charges. The stored state can be reloaded whenever the student wants. This button is always enabled also after the *Due Date*.
- **Show Answers:** This button is not used for our simulations. It is embedded by edX and we did not find a way of hiding it.

6.1.4. Different states of the running simulation

Figure 6.2 top shows the simulation after loading the course page but with the simulation not started. At this point the hidden charges are not placed. The field-lines are straight without bending. Figure 6.2 bottom shows the running simulation with our test set of the hidden charges as described above.. The filed-lines are bended through the influence of the placed hidden charges.

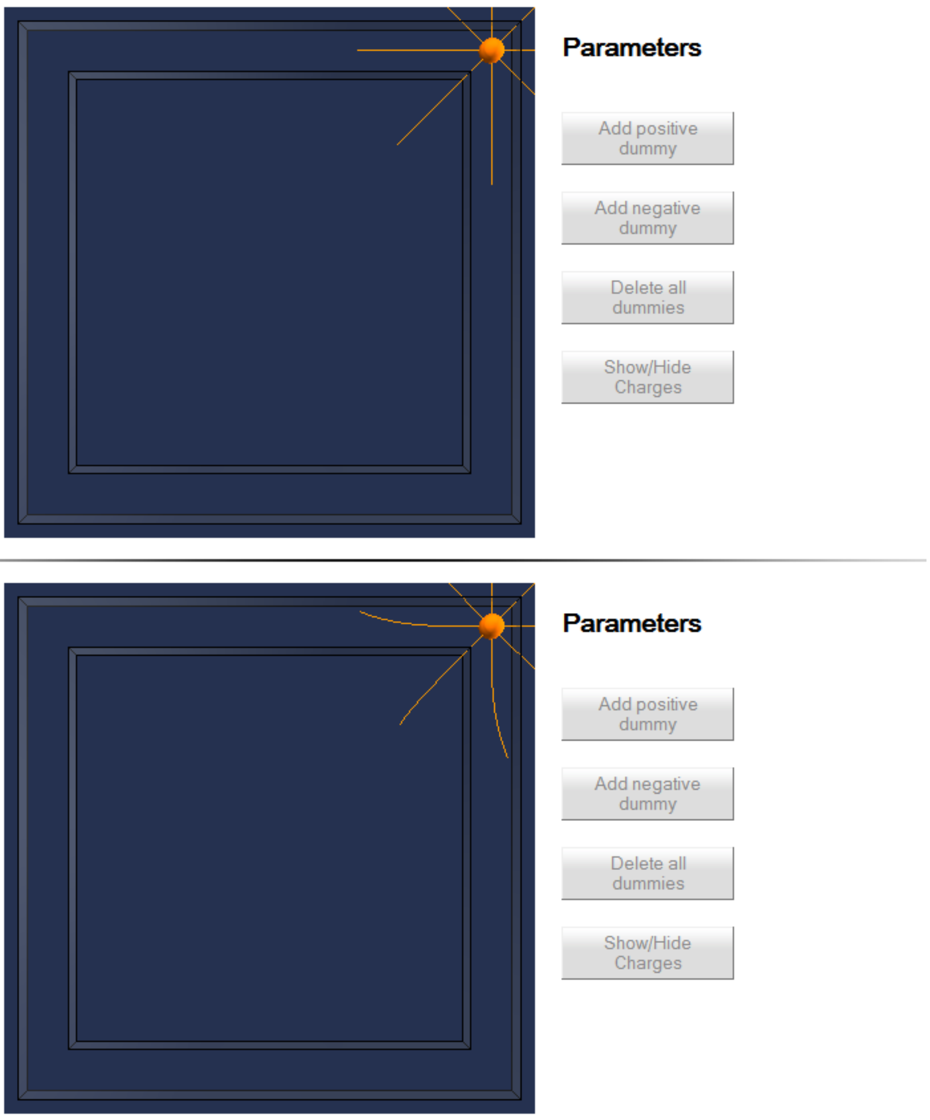
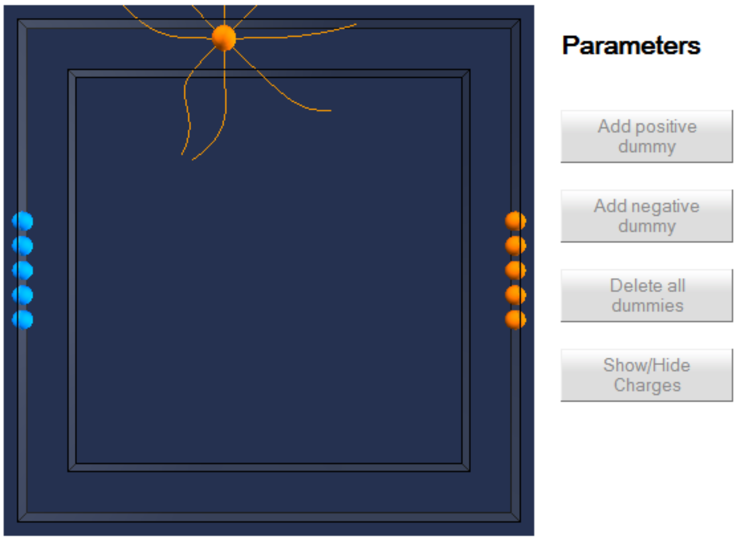


Figure 6.2.: Top: The simulation after page was loaded
Bottom: The simulation started

Figure 6.3 shows our test set for the hidden charges once with hidden and once with visible charges. At the top simulation the dummy charges are placed on the left and right side of the simulation. This are the positions where the dummies placed when added.



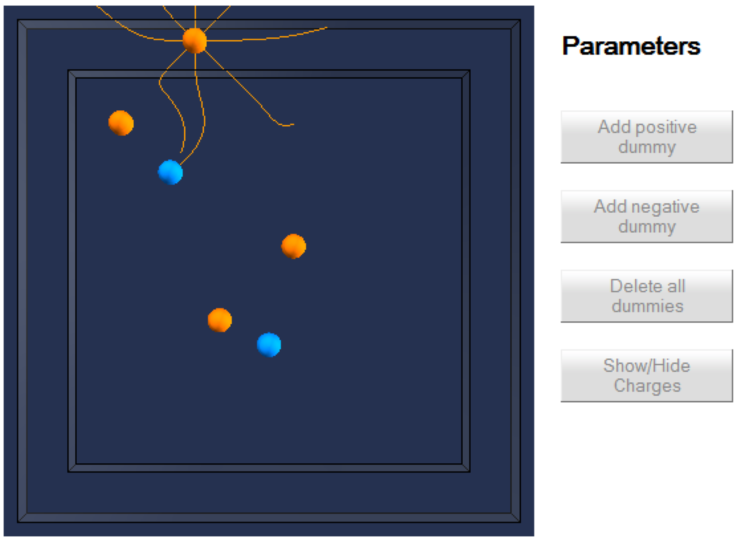


Figure 6.3.: Top: Running simulation with hidden charges and placed dummies on the sides. Bottom: Running simulation with hidden charged shown. This was our test set for the hidden charges.

Figure 6.4 shows a position of a hidden charge that is very hard to find as it is near the center and the field-lines are not touching the it. Important: The horizontal field line is not directly pointing to the center of the positive hidden charge. This is because of the influence of the other charges. On an exam course with Show/Hide button disabled this charge would be very difficult to find. But it also illustrates very well the influence of all charges on the field-lines.

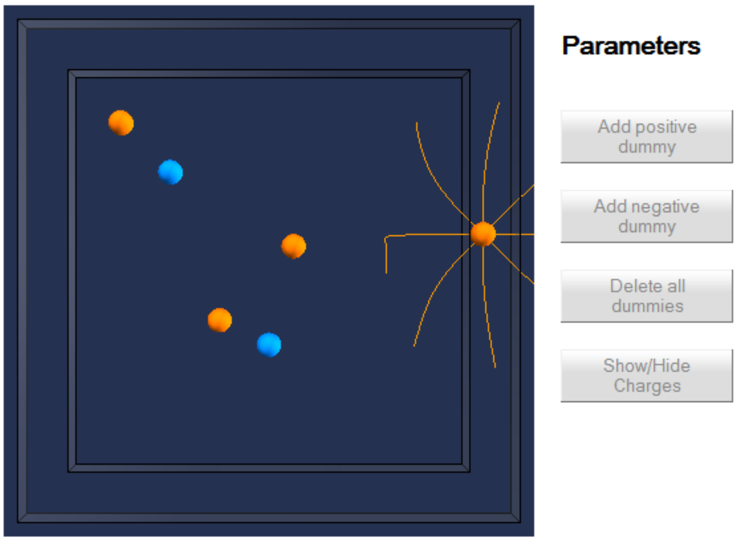


Figure 6.4.: Example for a hard to find position of a hidden charge.

6.1.5. The prototype of Point Charges Extended

At the end of the section about the first extended simulation we want to present a prior version we created and which problems we ran into. The prototype is shown in figure 6.5.

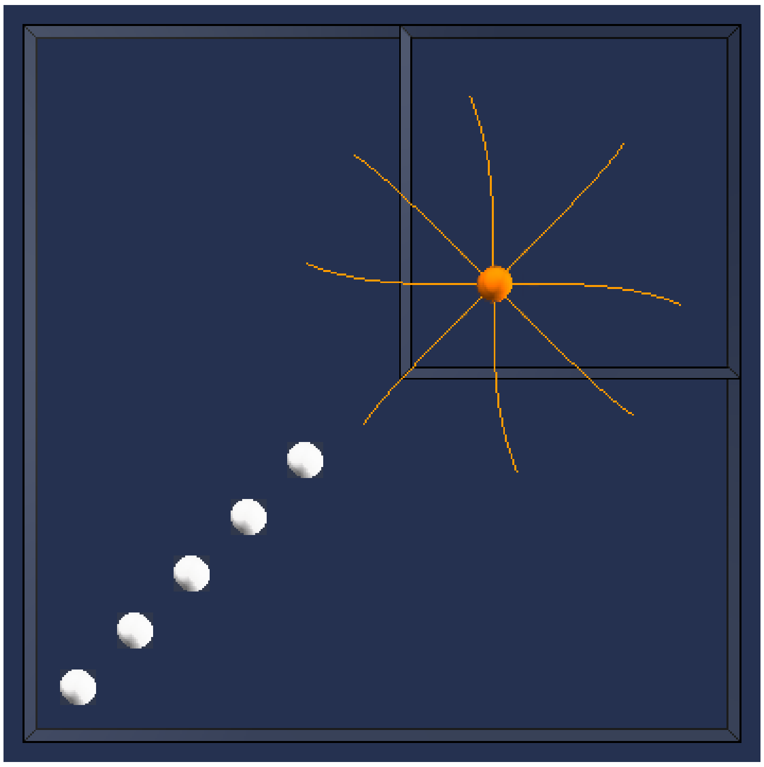


Figure 6.5.: A prototype of the Point Charges Extended simulation.

The idea was the same, find hidden charges with one moving charge. But the area where the visible charge could be moved was the upper right quarter as seen in the figure.

It is easy to see that with this setup, hidden charges placed at the positions of the white marker charges in the simulation can not be found exactly by moving the visible charge only in the small box. Furthermore hidden charges in the whole lower left quarter would be nearly impossible to find.

Another idea was to make a box of the size of the upper half of the simulation and place the hidden charges in the lower half.

6.2. The second TEAL simulation: Gausses Law Extended

The second TEAL simulation we extended was the *Gausses Law Simulation*. It was made to illustrate the Gausses Law on a Gaussian surface next to point charges. The Gaussian surface can be either a sphere or cylinder and is moved by sliders on the right side of the simulation, the charges are moved by dragging. Yellow arrows display the local electric field on the surface.

6.2.1. Requirements

When the student starts the simulation a number of hidden charges should be placed in the simulation as well as the Gaussian surface. The hidden charges should now be found by moving the Gaussian surface. In this simulation the student can place point charges. The assignment is to neutralize the hidden charges by placing the opposite charge over a hidden charge.

The yellow arrows should be by default scaled by the magnitude of the electrical field. That means, when an opposite charges is placed on a hidden charge, the field turns zero.

The next points are very similar to the requirements of our first simulation. Hence we will only give a short review. For more details see chapter 6.1.1.

When the student checks his answer, the positions and charges he placed are sent to the edX XML course file where they are graded by Python code.

The number of hidden charges should be determined by the course creator using python and should be at maximum five.

The positions of the initial charges should be created within the python code of the XML course file.

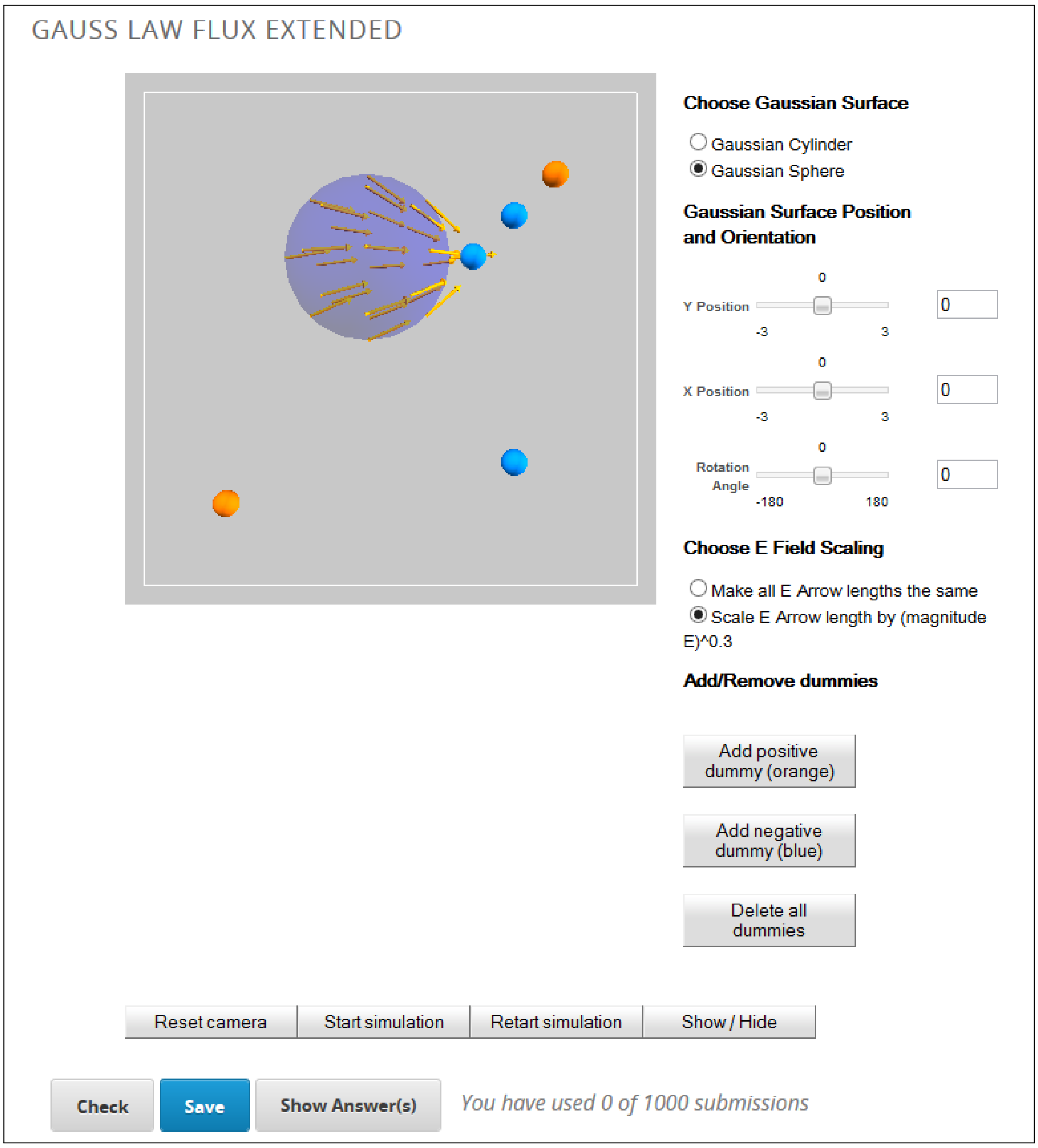
The position of the placed charges has to follow a grid.

The grading should be done by python in the XML file.

A Show/Hide Button should be implemented which shows and hides the hidden charges.

6.2.2. The simulation

Figure 6.6 shows the final *Gausses Law Extended* - Simulation. It shows the running simulation with the Gaussian surface and 5 point charges which are visible *hidden charges*. It shows the our test set for this simulation.


GAUSS LAW FLUX EXTENDED

Choose Gaussian Surface

Gaussian Cylinder [✓ Gaussian Sphere]

Gaussian Surface Position and Orientation

Y Position: 0 (range -3 to 3)

X Position: 0 (range -3 to 3)

Rotation Angle: 0 (range -180 to 180)

Choose E Field Scaling

Make all E Arrow lengths the same [✓ Scale E Arrow length by (magnitude E)^0.3]

Add/Remove dummies

You have used 0 of 1000 submissions

Figure 6.6.: The final *Gausses Law Extended* - Simulation

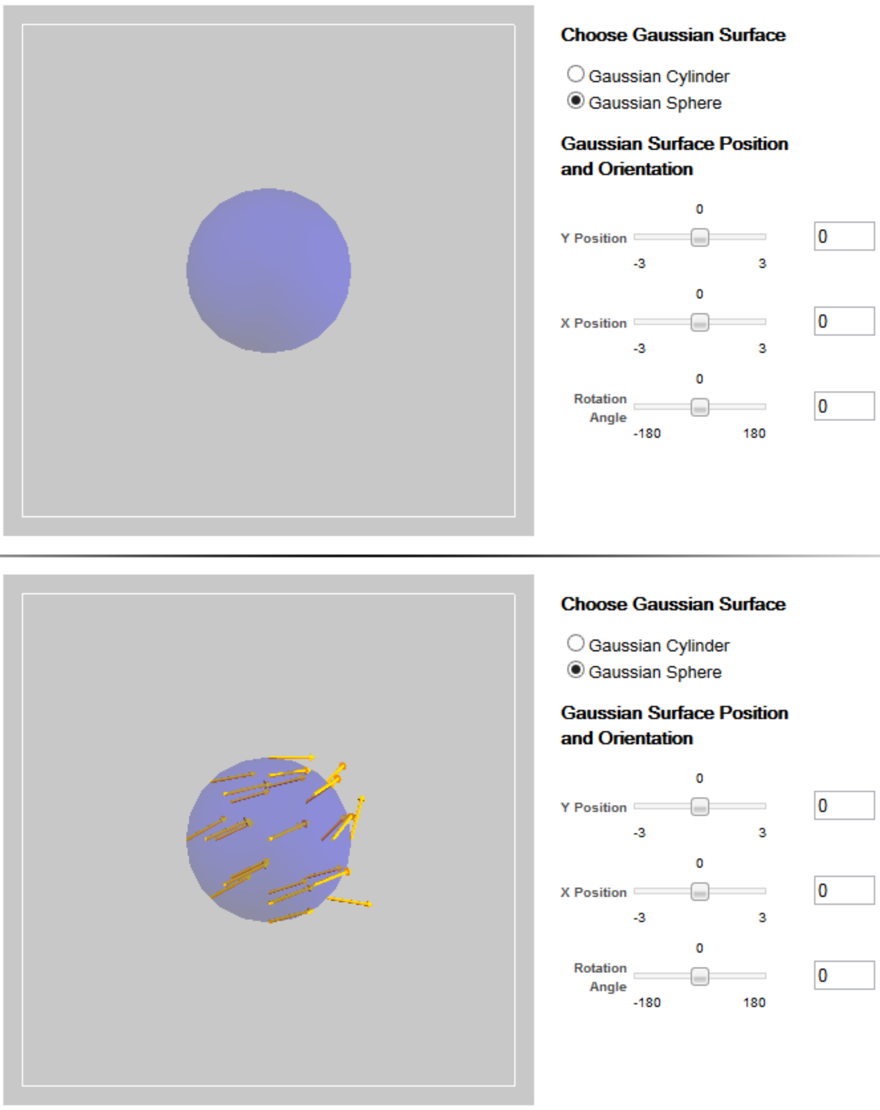
6.2.3. The buttons in detail

Some of the buttons are similar to the Point Charges simulation. Their description can be found in chapter 6.1.3. We will describe here only the different buttons and sliders.

- **Choose Gaussian Surface:** For switching the Gaussian surface between a cylinder and a sphere. By default the sphere is chosen.
- **Restart Simulation:** Loads and places the hidden charges, reloads previous set dummy charges and enables the 4 dummy buttons.
- **Three sliders:** They are used for moving the Gaussian surface within the simulation and rotate it, where rotating is only useful with the cylinder.
- **Choose E field Scaling:** Here the student can switch between scaled length and equal length of the yellow arrows. Sometimes the field can be very small and hence the arrows are very short. With the equal length setting the arrows are always well seen. This mode had to be adapted by us because with the equal length setting the arrows were always shown. So we had to display them when the field is zero.
- **Reset camera:** The simulation can be rotated. This is sometimes very helpful for finding the exact place of a hidden charge.

6.2.4. Different states of the running simulation

Figure 6.7 top shows the simulation after loading the course page but with the simulation not started. At this point the hidden charges are not placed and as the field is zero the arrows are not visible. Figure 6.7 bottom shows the running simulation. The hidden charges are placed in our test set as described above.


Choose Gaussian Surface

Gaussian Cylinder [✓ Gaussian Sphere]

Gaussian Surface Position and Orientation

Y Position	-3 … 0 … 3	0
X Position	-3 … 0 … 3	0
Rotation Angle	-180 … 0 … 180	0

Choose Gaussian Surface

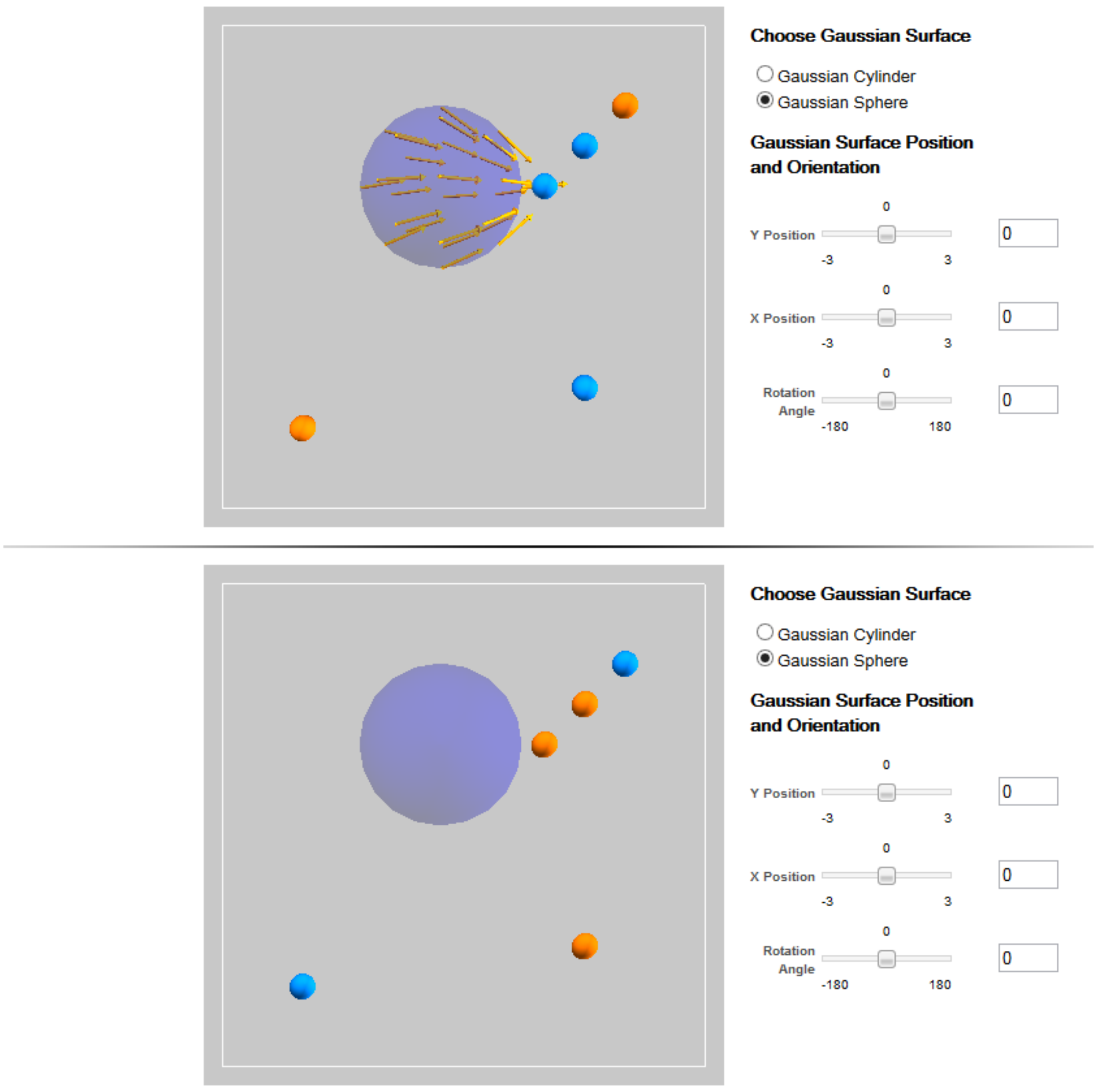
Gaussian Cylinder [✓ Gaussian Sphere]

Gaussian Surface Position and Orientation

Y Position	-3 … 0 … 3	0
X Position	-3 … 0 … 3	0
Rotation Angle	-180 … 0 … 180	0

Figure 6.7.: Top: The simulation after page was loaded
Bottom: The simulation started

Figure 6.8 shows at the top our test set for the hidden charges with visible *hidden charges*. At the bottom charges where placed correctly over the hidden charges and the field got zero.



Choose Gaussian Surface

Gaussian Cylinder [✓ Gaussian Sphere]

Gaussian Surface Position and Orientation

Y Position 0 (-3 to 3): 0

X Position 0 (-3 to 3): 0

Rotation Angle 0 (-180 to 180): 0

Choose Gaussian Surface

Gaussian Cylinder [✓ Gaussian Sphere]

Gaussian Surface Position and Orientation

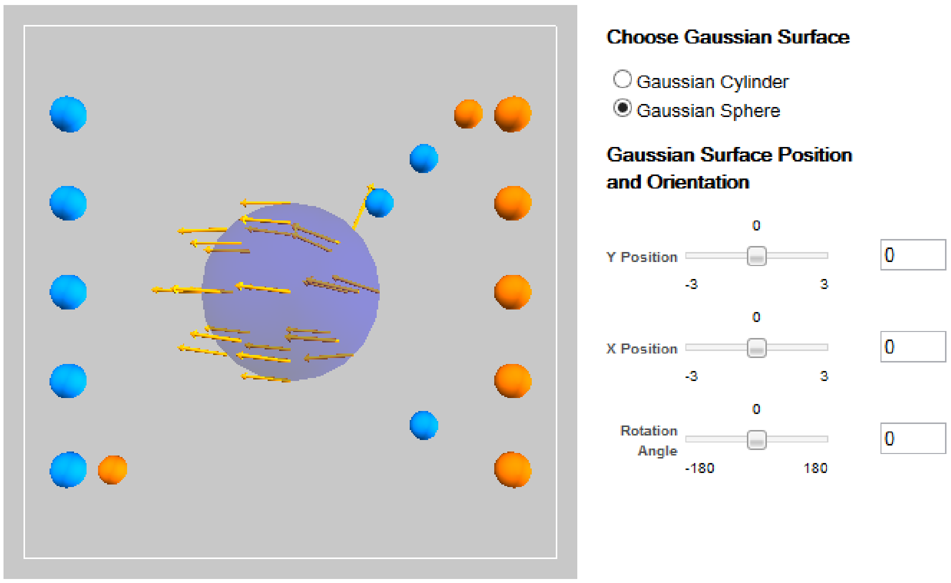
Y Position 0 (-3 to 3): 0

X Position 0 (-3 to 3): 0

Rotation Angle 0 (-180 to 180): 0

Figure 6.8.: Top: Running simulation with visible *hidden charges* in our test configuration. Bottom: Running simulation with correct placed charges over the hidden charges.

The last figure 6.9 we want to present from our second extended simulation shows the visible *hidden charges* as well as all possible charges, placed by a student. It is clearly recognizable that the students places charges in contrast to our first extended simulation where a student could place dummies. Furthermore one can see that the placeable charges are a little larger to distinguish them from the *hidden charges*.



Choose Gaussian Surface

◯ Gaussian Cylinder
◉ Gaussian Sphere

Gaussian Surface Position and Orientation

Y Position	-3 … 0 … 3	0
X Position	-3 … 0 … 3	0
Rotation Angle	-180 … 0 … 180	0

Figure 6.9.: Placed charges and their influence on the field arrows.

6.3. The third TEAL simulation: Amperes Law Extended

The third simulation we extended was the Amperes Law simulation. It demonstrates the impact of line currents on a open Amperean surface which could be either a circle or rectangle. The line currents can be moved by dragging with the mouse, the Amperean surface with sliders next to the simulation.

6.3.1. Requirements

When starting the simulation a number of hidden *line currents* are placed within the simulation. They should be found by the students moving the Amperean surface. We used the system from our second simulation again, where the student has to eliminate all hidden line currents by placing opposite currents on top of them.

The next points are very similar to the requirements of our first and second simulation. Hence we will only give a short review. For more details see chapter 6.1.1 and 6.2.1.

The blue arrows should be by default scaled by the magnitude of the field.

When checking the answer the positions of the placed line currents are sent to the edX XML Course file where they are graded by a Python function.

The number of hidden line currents is set by the course creator.

The initial positions of the hidden line currents are set by the course creator using Python.

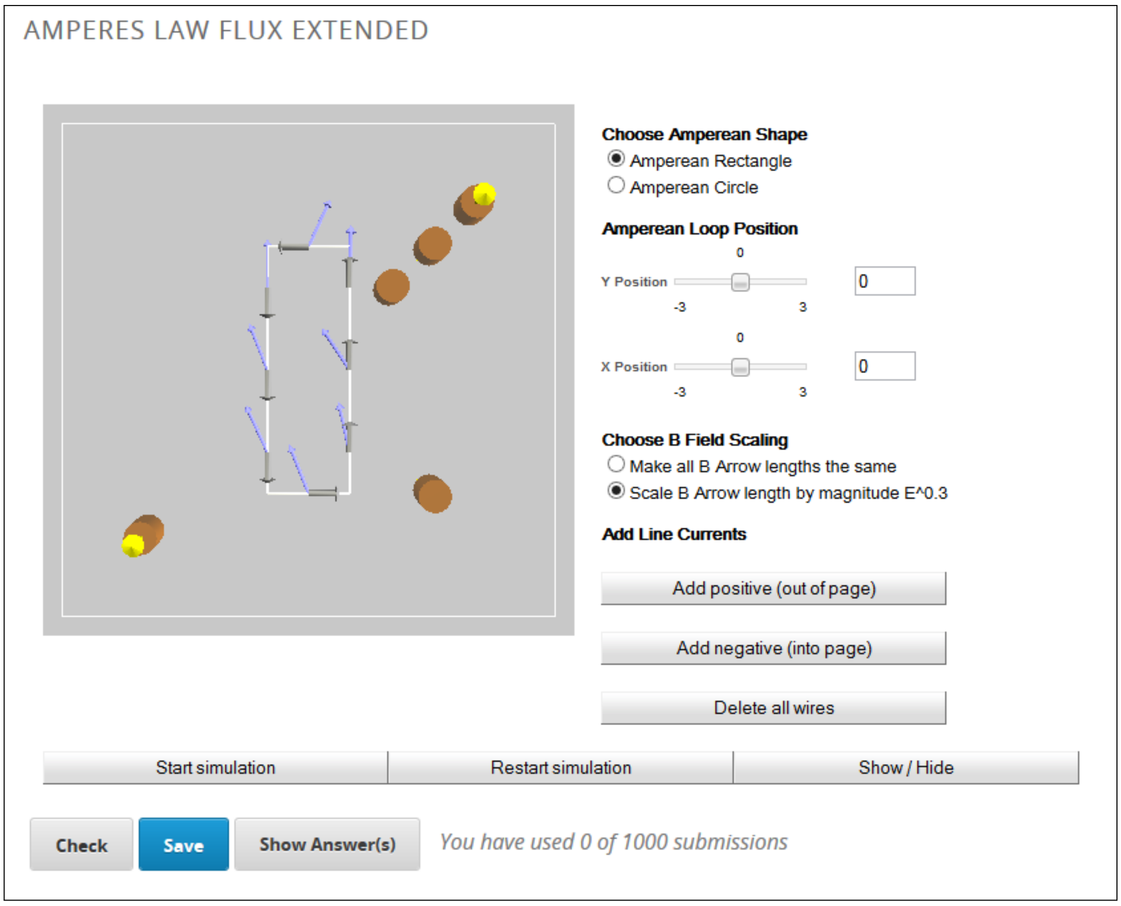
The student can place the line currents only on a grid.

Grading should be done by the edX XML File using Python.

A Show/Hide button should be implemented.

6.3.2. The simulation

Figure 6.10 shows the final *Amperes Law Extended* - Simulation. It shows the *Amperean Rectangle* as well as 5 hidden *line currents* which are enabled by the *Show/Hide* button. The image shows the test set for this simulation.


AMPERES LAW FLUX EXTENDED

Choose Amperean Shape
[✓ Amperean Rectangle]
Amperean Circle

Amperean Loop Position
Y Position: 0 (range -3 to 3)
X Position: 0 (range -3 to 3)

Choose B Field Scaling
Make all B Arrow lengths the same
[✓ Scale B Arrow length by magnitude E^0.3]

Add Line Currents

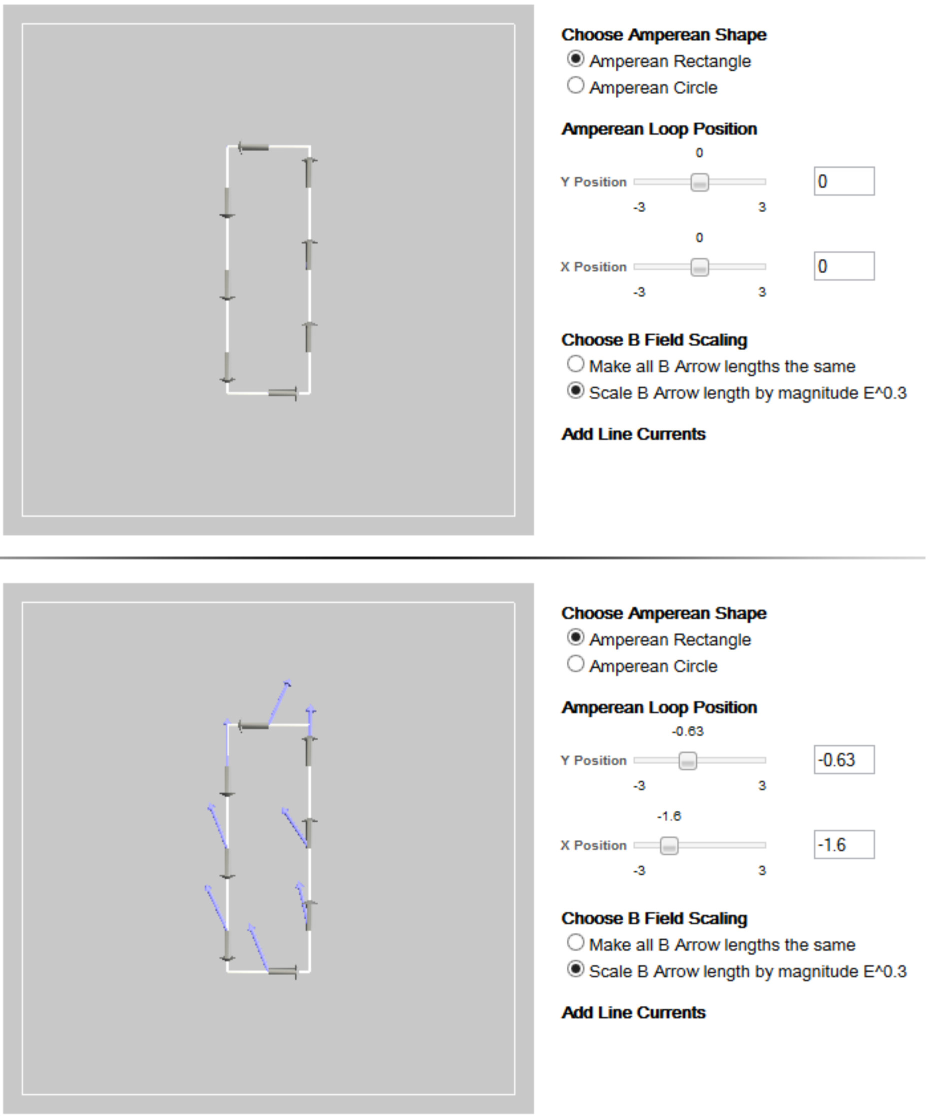
You have used 0 of 1000 submissions

Figure 6.10.: The final *Amperes Law Extended* - Simulation

The buttons have the same function as in our first and second simulation. For more details see section 6.1.3 and 6.2.3.

6.3.3. Different states of the running simulation

Figure 6.11 top shows the simulation after loading the page. The simulation is not running. Bottom shows the the simulation running with the hidden line currents at our test set, as described in figure 6.10.



Choose Amperean Shape

[✓ Amperean Rectangle]
Amperean Circle

Amperean Loop Position

Y Position: 0 (-3 to 3) [0]

X Position: 0 (-3 to 3) [0]

Choose B Field Scaling

Make all B Arrow lengths the same
[✓ Scale B Arrow length by magnitude E^0.3]

Add Line Currents

Choose Amperean Shape

[✓ Amperean Rectangle]
Amperean Circle

Amperean Loop Position

Y Position: -0.63 (-3 to 3) [-0.63]

X Position: -1.6 (-3 to 3) [-1.6]

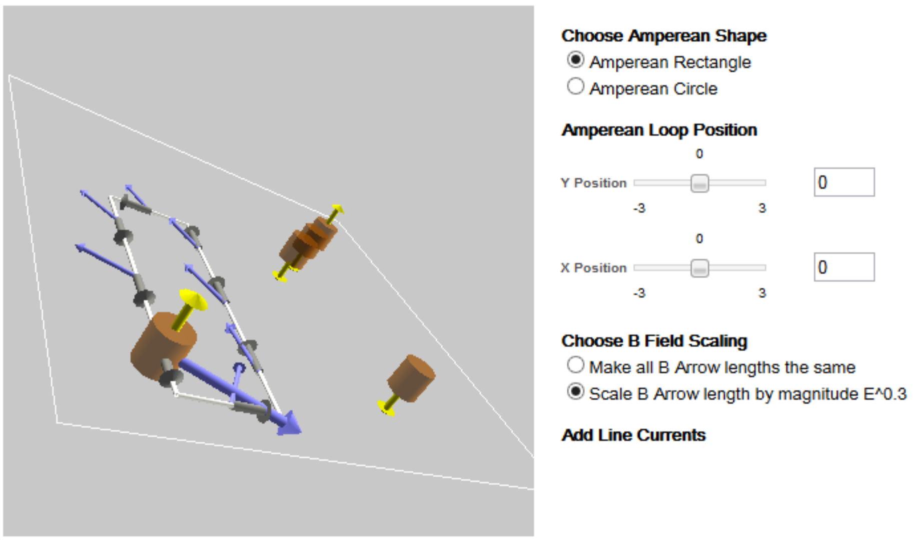
Choose B Field Scaling

Make all B Arrow lengths the same
[✓ Scale B Arrow length by magnitude E^0.3]

Add Line Currents

Figure 6.11.: Top: The simulation after page was loaded
Bottom: The simulation started

Figure 6.12 shows the result when an opposite line current is placed on top of a hidden line current. The field gets zero and the blue arrows disappear.

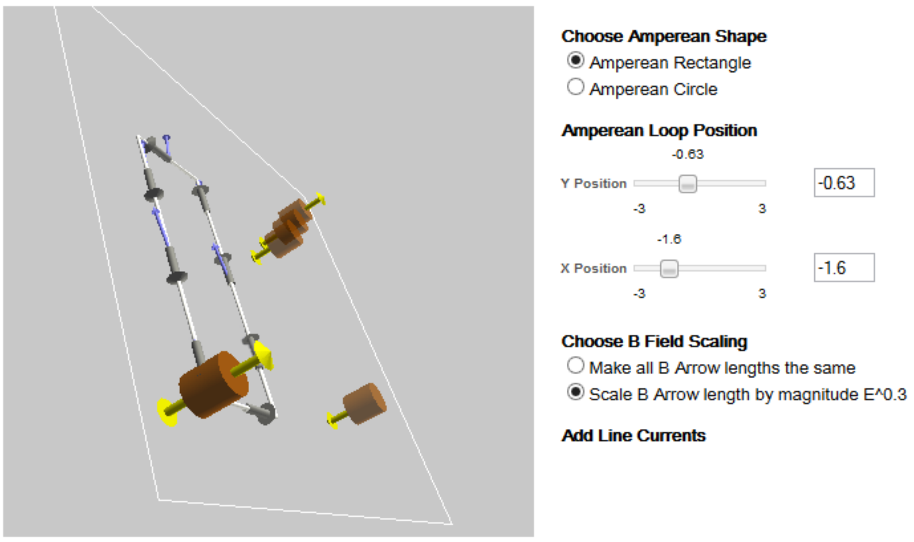


Choose Amperean Shape
[✓ Amperean Rectangle]
Amperean Circle

Amperean Loop Position
Y Position: 0 (slider -3 to 3)
X Position: 0 (slider -3 to 3)

Choose B Field Scaling
Make all B Arrow lengths the same
[✓ Scale B Arrow length by magnitude E^0.3]

Add Line Currents



Choose Amperean Shape
[✓ Amperean Rectangle]
Amperean Circle

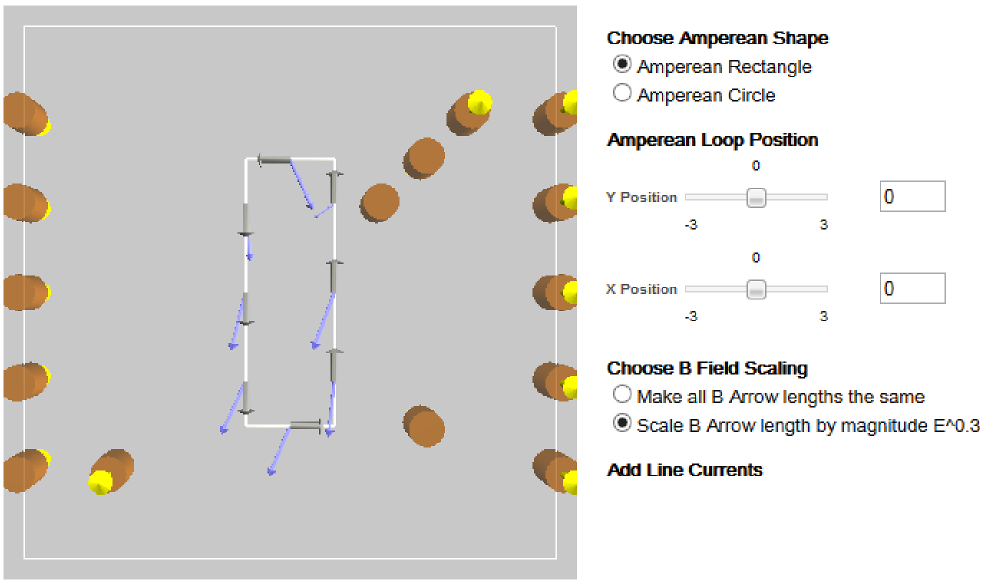
Amperean Loop Position
Y Position: -0.63 (slider -3 to 3)
X Position: -1.6 (slider -3 to 3)

Choose B Field Scaling
Make all B Arrow lengths the same
[✓ Scale B Arrow length by magnitude E^0.3]

Add Line Currents

Figure 6.12.: Top: Field induced by a line current.
Bottom: An opposite line current placed on top of the first current. The field gots zero.

The last figure 6.13 we want to present from our third extended simulation shows the placed line currents and their impact on the field.



Choose Amperean Shape

(●) Amperean Rectangle

( ) Amperean Circle

Amperean Loop Position

Y Position: 0 (slider -3 to 3) [0]

X Position: 0 (slider -3 to 3) [0]

Choose B Field Scaling

( ) Make all B Arrow lengths the same

(●) Scale B Arrow length by magnitude E^0.3

Add Line Currents

Figure 6.13.: Simulation with all line currents placed.

6.4. Discoveries

During our testing with random positions for the hidden elements in our simulations we recognized early that this topic needs more attention then we thought it would need. We wrote about this shortly in the section about starting our extended simulations with random values 5.2. For example in figure 6.3 we showed a very easy to find hidden charge of the point charges extended simulation. In contrast in figure 6.4 a very hard to find hidden charge is shown. Before our tests we thought we can place the hidden elements randomly within the simulation maybe with some space to the outer border. Already the first test showed us that this assumption was wrong. By placing the hidden elements complete randomly the assignments for the students would be completely different hard to solve. The main reason for placing hidden elements randomly was to prevent students from cheating. To solve the problem of different hard to solve problem sets we had to limit the possible positions the hidden charges could be placed.
One idea to give same difficult assignments but vary the problem set is to only randomize only the sign of the charges for every student. But we think it would be too easy for the students to figure out that the positions are same and only the charges are different.
The final solution was to create a number of similar difficult problem sets, about 20, and choose for every student randomly one of this problem sets. Additionally the first idea could applied and the charges could be shuffled to create more different problem sets.

An idea for teaching was to create more problem sets getting harder after each other or create sets to show special problems. These learning sets could be with enabled Show/Hide button so the student can easily check the positions. After the learning sets the exam set is with disabled Show/Hide button. For example starting with one hidden element what should be easy to find. And with every problem set another hidden element is added. Also special cases can be implemented for example with very narrow placed hidden elements to show the influence of them to each other and the field. After some of this learning sets the student should be well prepared for the exam set.

6.5. Summary – The incidental gamification of the TEAL simulations

In this chapter we described our three extended simulations:

- Point Charges Extended 6.1
- Gausses Law Extended 6.2
- Amperes Law Extended 6.3

We chose this three simulations as they are used in the first weeks of the course 8.02x. Fortunately from the programmers point of view these simulations were similarly to each other so we could reuse a lot of our code. Otherwise we would definitely not have been able to extend three simulations.

As we described the technical background and the interface in the previous section 5 we focused in this chapter on the simulations themselves, on the requirements, the results and solutions as well as the handling.

The prime requirement for all simulations was that the extended simulations can be used for answering questions asked by the edX course. The idea for this three simulations was to place hidden elements in the simulation which have to be found by the student. This should make the student use the simulation on the one hand, on the other hand should the work with the simulation make the student understand the physical background behind the simulation.

The requirements in detail:

- When starting the simulation, hidden elements should be placed
- The number of hidden elements should be determined by the course creator within the edX XML course file. The maximum should be 5 elements.
- The positions of the hidden elements should be defined by the course creator within the edX XML course file.
- The student can place dummy elements for marking his found positions.
- The dummy elements should be placeable on a grid.
- Grading should be done by Python within the edX XML course file.
- A *Show/Hide* button should be implemented which can be enabled and disabled by the course creator for every problem set,

The incidental gamification

During our work we presented our extended simulations to different people to get feedback as well as for testing and finding bugs. What we recognized was the fact, that almost all of them talked about our extended simulations as games and about playing them. In section 2.2 of this work we discussed the term *gamification*. Referred to the described definition of gamification our extended simulations fulfill two points:

1. Challenge - During the learning sets the user of our extended simulations can check his answer and gets instantly feedback if he was successful. The user is not starting the simulation to learn the physics behind them. He starts the simulation to find the correct positions of the hidden elements and that is a challenge.
2. Fun - Almost every person who tested our simulations had fun using it.

That does not mean we created games but it shows that our extended simulations fulfill some features of games. We hope the students will enjoy the extended simulations as our testers did. And we hope they will learn easier and faster using them.

7. Evaluation

Because the use of our new simulations in the MIT physics course 8.02x was postponed, the evaluation of the extended simulations by MIT students was also postponed. There will be an evaluation of our simulations in the next weeks by MIT and TU-Graz colleagues for completion our work. We are sorry that this will need some longer and we could not insert here.

At this point we can just refer to the mid-semester evaluation [12] about the course in general.

8. Future Work

Professor Belcher is a visionary. He always tried to find new learning methods and additional experiments to help students learning the topics of the basic physics courses. That started with experiments in front of webcams, went to the virtual TEAL simulations and came to the online platform edX. And this work is part of the evolution of the basic physics courses at the MIT.

8.1. Near future - The rest of the seven TEAL simulations

The next step will be to extend the rest of the seven TEAL simulations which where ported to JavaScript. During this master thesis we also worked on the rest of the simulations and how they could be used for interactive questions. Indeed we were always sure that six month are way too short to extend all of the seven TEAL simulations, but we did it also to know which requirements our interface will have to fulfill. During the entire development of the interface we tried always to keep it as open and universal as possible to make sure it can be used for future work.

At this point we want to introduce the other four simulations and the ideas we think would be on the one hand good for education and on the other hand realistic to implement from the programmers point of view.

It is important to keep in mind that this are only ideas which where not tested. As our work with the three extended simulations has shown, between the first ideas and the final simulations sometimes is a huge difference. The reasons can be different. We had once the problem that an assignment was simply unsolvable with the tested implementation. Another idea was for us programmers good but from the educational point of view it was senseless. And for other ideas we just thought, lets try it out and see what colleagues think.

We think the following ideas could be work very well and lead to good extended simulations. We will give them a try in future but of course we can not be sure if they will become useful extended simulations or if they will be implemented completely different.

Floating Coil

In this simulation a current carrying coil rests on a platform centered on the axis of a permanent magnet. With the slider on the right the current in the coil can be set which can pull the ring towards the magnet or push it away from the magnet. The simulation is shown in figure 8.1.

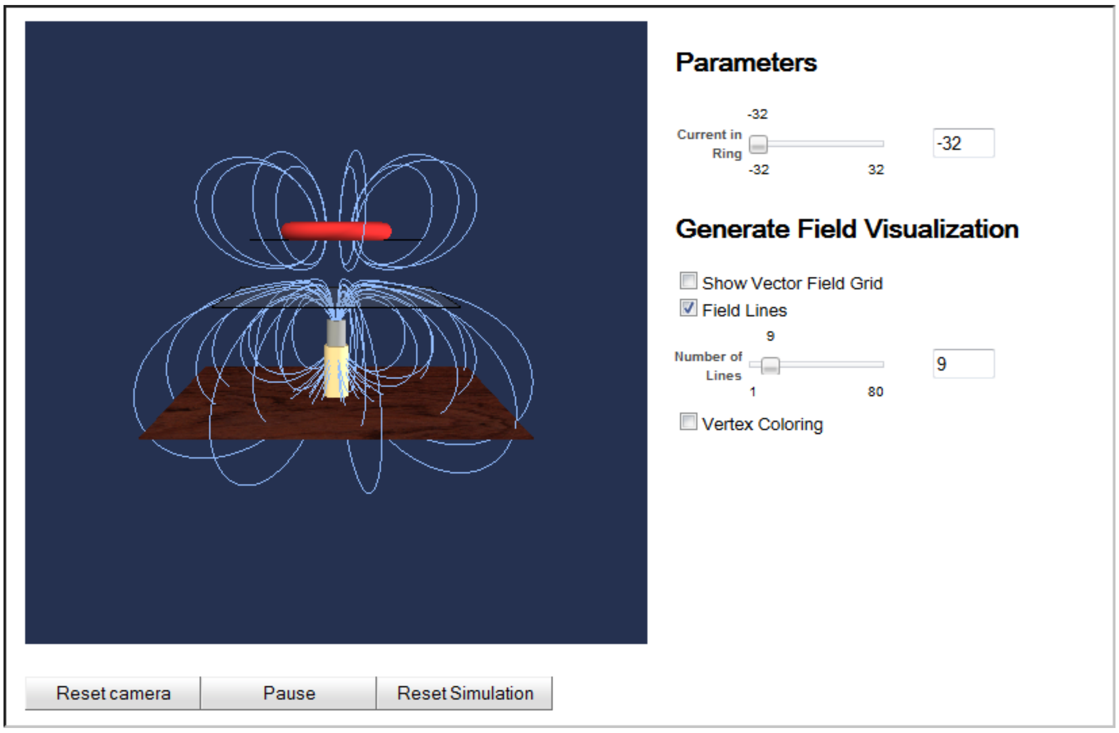


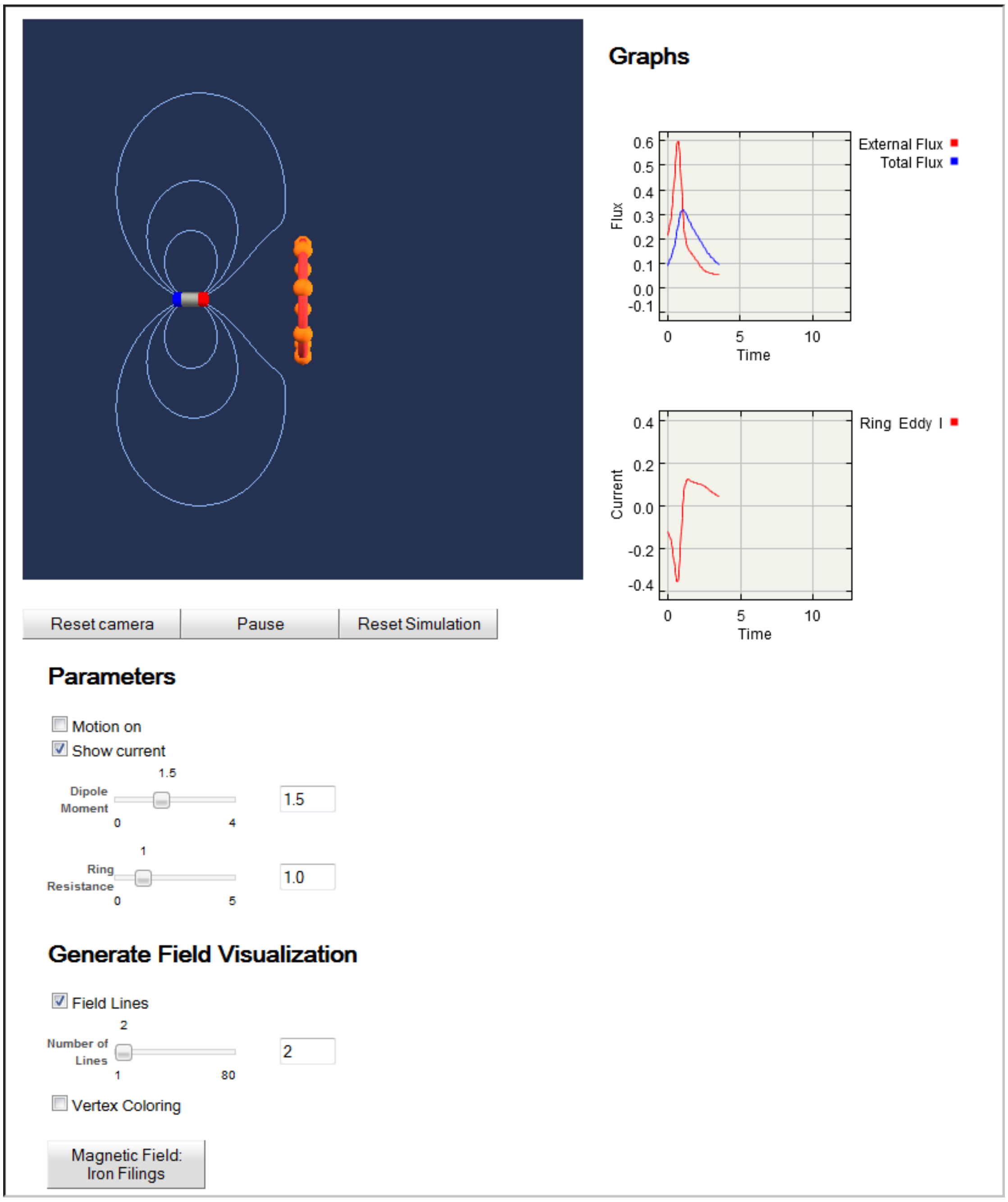
Figure 8.1.: The Floating Coil simulation

One idea was to randomly rotate the magnet, with the north pole or the south pole at the top. The student has then to find out which pole is at the top depending on which current he has to choose, a positive or negative one.

Another question could be which current is used to elevate the ring above a given hight. Therefor a hight-meter would be implemented, what could be a single text box.

Faraday's Law

In this simulation a magnet is movable on the axis of a ring with a self-inductance and a resistance. I tis shown in figure 8.2.



Parameters

☐ Motion on
☑ Show current

Dipole Moment: 1.5 (slider 0 – 4)

Ring Resistance: 1.0 (slider 0 – 5)

Generate Field Visualization

☑ Field Lines

Number of Lines: 2 (slider 1 – 80)

☐ Vertex Coloring

Figure 8.2.: The Faraday's Law simulation

When moving the magnet along is axis through the ring, the ring will start rotating. The rotation speed depends on the speed of the movement of the magnet. Tho rotation direction depends on the direction of the movement of the magnet. Some different states of the simulation can be seen in figure 8.3.

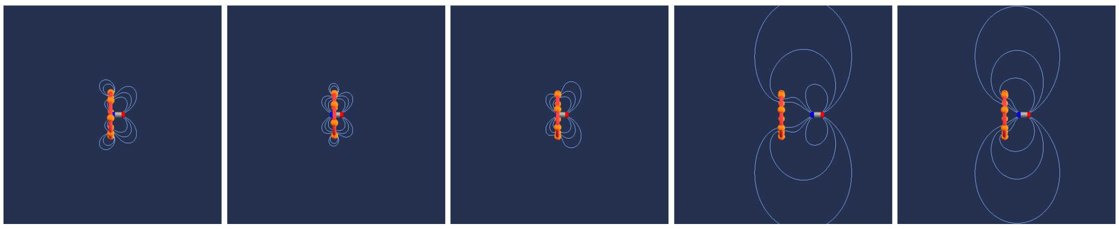


Figure 8.3.: A series of the Faraday's Law simulation

A further option is to let the magnet automatically move sinusoidally through the ring what lead to the sinus-shaped curves which can be seen in figure 8.4 on the right.

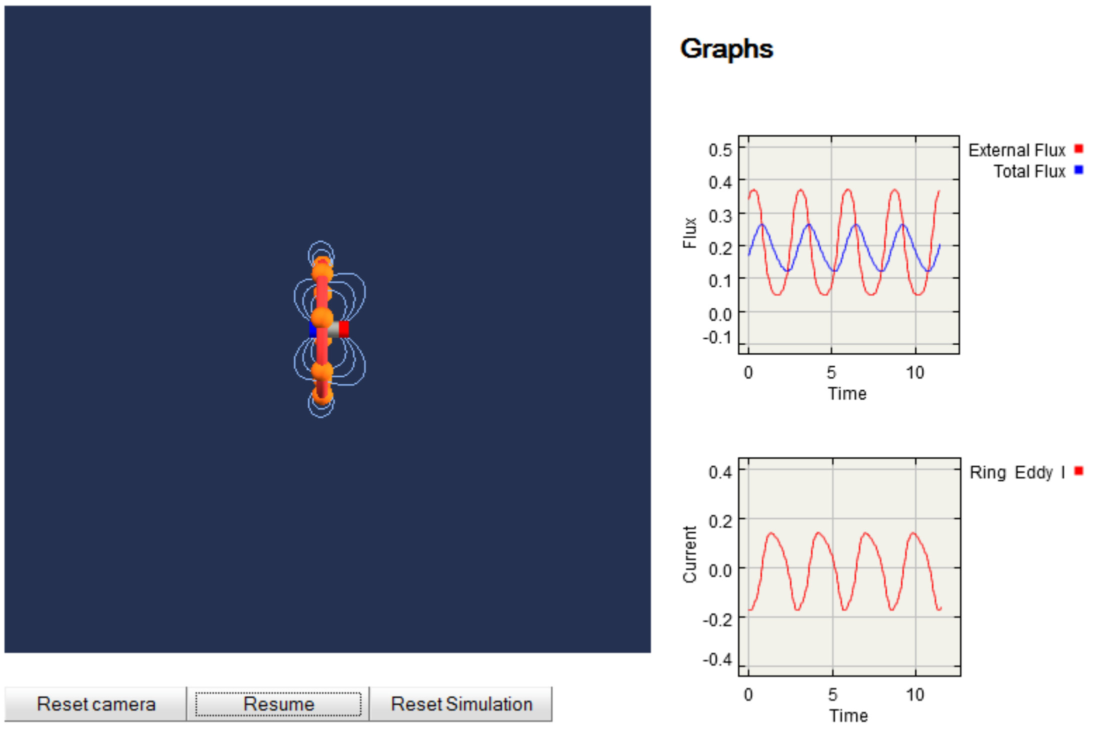


Figure 8.4.: The Faraday's Law simulation with sinusoidally moving magnet.

Questions for this simulation could be to give the student a screen-shot of the flux and current curves showing sinus-shaped waves. The student has then to set the right *Dipole Moment* and *Ring Resistance* to obtain the same curves as questioned.

Furthermore we could implement new sliders for the movement of the magnet, one for the speed and one for each endpoint of the magnet. The student should then again set this sliders to get the same curves as questioned.

Charge by Induction

Figure 8.5 shows the *Charge by Induction* simulation. With the slider on the right the current of the big charge in the middle can be set. Between the cylinders are a number of point charges. Furthermore two other visualizations can be displayed as shown in figure 8.6.

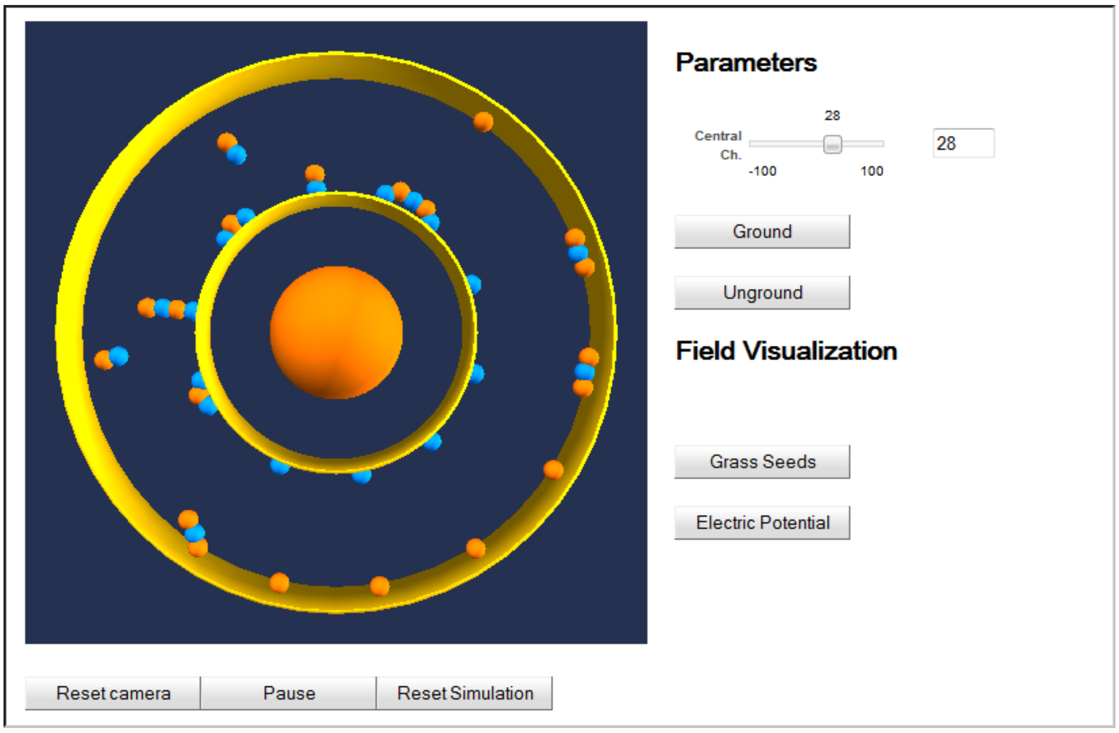


Figure 8.5.: The Charge by Induction simulation.

Figure 8.6 shows one state of the simulation in all three possible visualizations.

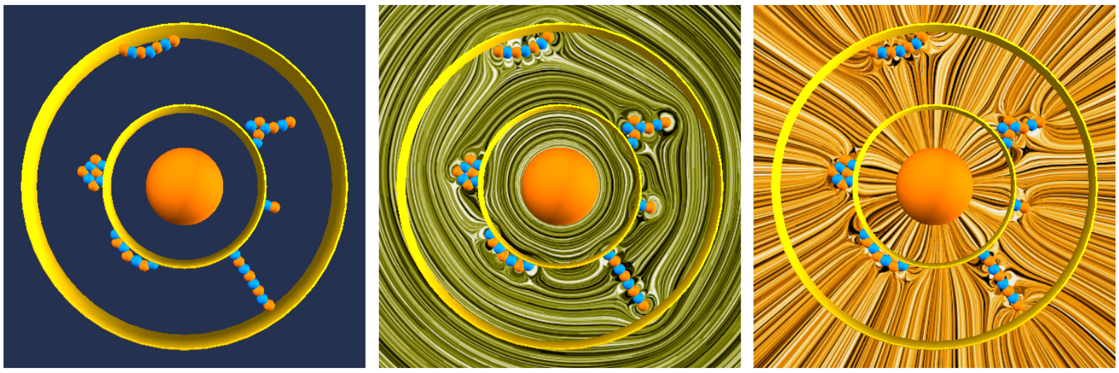


Figure 8.6.: Different visualizations of the Charge by Induction simulation.

One idea for this simulation was that the course creator can place point charges, as many as he wants and where he wants. Then the two visualizations are stored. The student gets this visualizations as background of his simulation and has to place point charges to accomplish the same field visualization.

For this idea again a number of different sets can be created and every student could get randomly one of them.

Plane Wave simulation

The simulation shows a pink sheet of positive electric charge. By shaking it plane electromagnetic waves are generated. The yellow arrows represent the radiation electric field, the blue arrows the radiation magnetic field.

The sheet can be moved up and down by the user, as seen in figure 8.7. The sheet can also be automatically moved up and down which can be seen in figure 8.8. The movement leads to sinus-shaped arrows.

The idea was to give the student a screen-shot of this simulation with sinus-shaped arrows of specific length. The student should then have sliders for the automatically moving of the sheet to obtain the same sinus curve. The student should have to set the direction of the movement (up/down and backward/forward) as well as the speed of each direction.

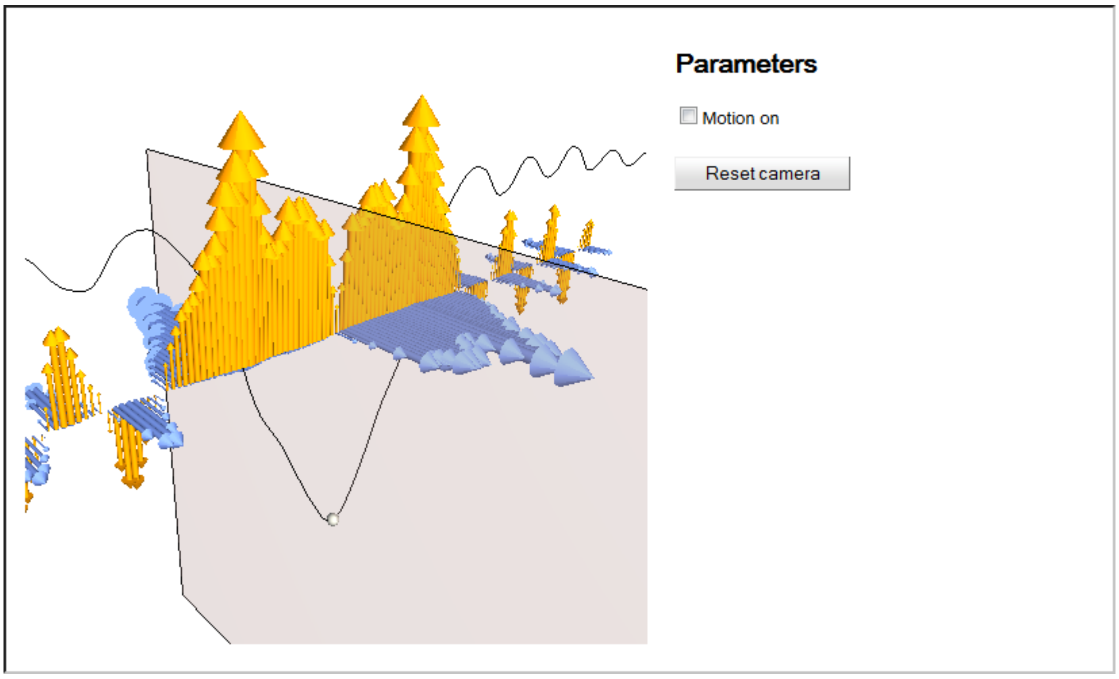


Figure 8.7.: The plane wave simulation.

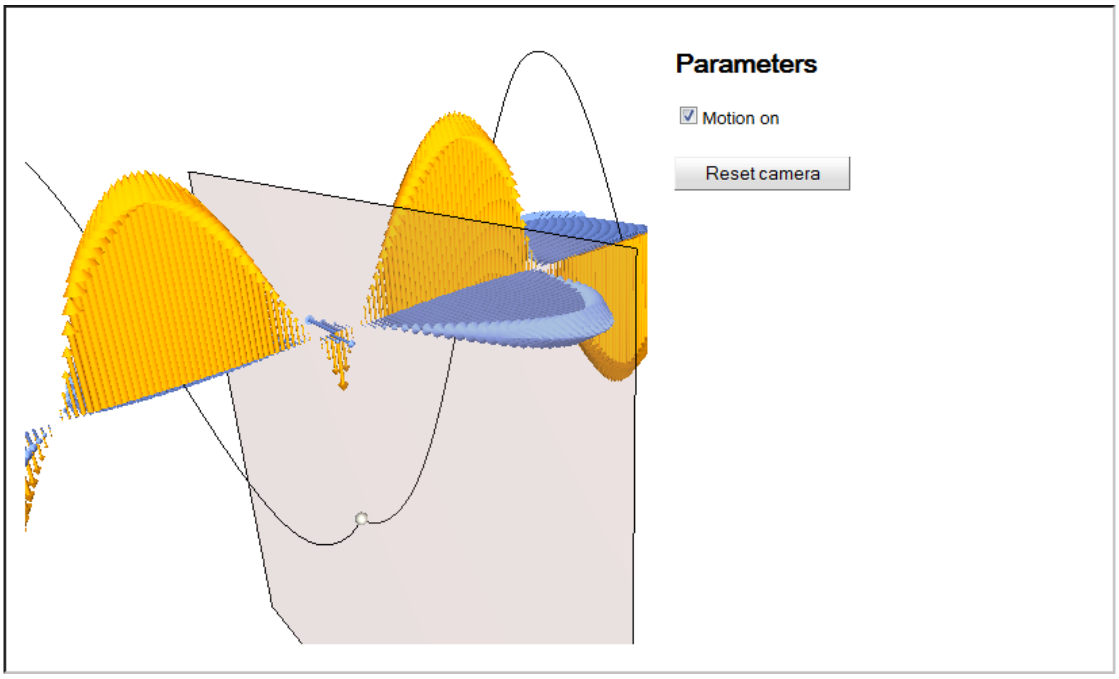


Figure 8.8.: The plane wave simulation.

8.2. Ultimate goal - A complete new framework for simulations

One of the disadvantages of the seven simulations which were ported to JavaScript is that everyone was ported on its own and from different people. That means there is code in every simulation written same. And so is our new code which was added for extending them. The reason for this circumstance was time as this way was the fastest and easiest. The physics courses with the simulations (and now the extended simulations) are still in development and evolution.

A huge benefit of e-learning and online courses is the detailed feedback from the students about every part and aspect of them. The seven simulations have shown that they can help students understanding physics but evaluations have shown that students did only insufficient make use of them. Our extended simulations are the next evolutionary step of the basic physics courses. And depending on future evaluations they will be improved again.

The logical next step will be to create a whole new framework for physics simulations. The programmers at CECI call it a player for the simulations. An environment where many different simulations can run and interact with an enveloped system like edX.

The benefits for such a player would be on the one hand an easier and faster implementation of new simulations as it would provide a broad library. This would help reusing code in all simulations and prevent of duplicated code. On the other hand it could implement a more stable and extensive interaction with edX. This could possibly be in cooperation with edX and hence it could be used from other institutes too.

8.3. Improvements on jsinput

One problem during our work was the documentation of jsinput. At the time this work was written, jsinput was still in development and changed through the last years. And so did the documentation. Thats why we found different versions of jsinput documentation and often we just had to try how the current version works. We were also in contact with the developer of jsinput and gave feedback on it and what we would need in future, so we know jsinput is further developed.

One thing we know for sure is that jsinput will get an mechanism to send initial values from edX to JavaScript applications. And we are sure there will be a lot of improvement in future of jsinput. Maybe in near future there is a new version of jsinput what would make our work a lot easier.

9. Summary

The basic physics courses at the MIT are not the favorite courses of students. For motivating students to learn and furthermore simplify learning the TEAL classroom was introduced. It offers different multimedia techniques as well as interactive experiments. As technology improved over the last years and the Internet got normal to almost everyone of us more and more parts of the physics courses where brought to an e-learning course using the online platform edX.

Part of this online courses where seven virtual TEAL simulations which where ported to JavaScript. Being more exact, the simulations where written in Java and compiled to JavaScript using GWT. The simulations should help students understand the learning topics.

Evaluations of previous courses have shown that students did not or just very short work with the simulations. So the idea was born to make them interactive and use them for answering questions through online homeworks and exams.

The first part of our work was to find out if it is possible to create an interface between edX and the simulations to achieve communication and interaction between them. From an edX programmer we got the hint to use the edX method *jsinput*.

Jsinput was made to provide a communication between the edX course, which is written in XML, and JavaScript applications. Unfortunately we had two problems using jsinput. First, our simulations where not written in JavaScript, instead they where written in Java and compiled to JavaScript. First tests show that we were not able to set up a stable communication, there were always synchronization problems. The second problem was the lag of sending initial values to the JavaScript application using JavaScript.

We solved this problems in two steps. For the synchronization problem we used the HTML host file of our JavaScript simulation as a buffer between edX and our Java simulation. That means, edX did not directly communicate with the Java written simulation. It was communicating only with the HTML host file. And the Java simulation was also communicating only with the HTML host file of the simulation. This led to a stable and reliable interaction.

For the initial values problem we had to create a work around. As we could not use jsinput we implemented a second communication using postmessages. This are messages sent by JavaScript between the edX XML course file and the HTML host file. Again, we did not directly communicate between edX and our Java simulation.

Once the technical problems where solved we started to extend the first simulation. We chose the Point Charges simulation as this would be the first simulation used during the next online course. At the beginning we had to think about how we can extend this simulation to ask students useful questions. The idea was to place hidden charges which has to be found by the students. Furthermore we wanted to place these hidden charges randomly to avoid cheating between students.

The second simulation we extended was the *Gauss Law* simulation and the third simulation was the *Amperes Law* simulation. Both extended simulations implement a similar approach to our first extended simulation using hidden elements which have to be found by the students.

During the work we had colleagues testing our extended simulations. The common opinion was that they liked to work with them and for all of them it was some sort of playing a game. So it seemed we accidentally gamified the simulations. In the past games have become increasingly usage in education. It has been shown that games and simulations can rise the motivation to learn. For us we think it is great if students have fun with the extended simulations as that means they are having fun at learning. And if we achieved that students like the basic physics course a little bit more we were successful. We are excitedly looking forward to the first evaluation of a course using our simulations.

List of Figures

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

List of Abbreviations

CECI	Center for Educational Computing Initiatives
e-learning	Electronic Learning
ETH Zürich	Eidgenössische Technische Hochschule Zürich
GWT	Google Web Toolkit
HTML	Hypertext Markup Language
JSNI	JavaScript Native Interface
MIT	Massachusetts Institute of Technology
MOOCs	Massive Open Online Courses
SDK	Software Development Kit
STEM	Science Technology Engineering and Mathematics
TEAL	Technology-Enabled Active Learning
TEALsim	Technology-Enabled Active Learning Simulation
TU Graz	Technical University of Graz
URL	Uniform Resource Locator
XML	Extensible Markup Language

Appendix A.

Code parts

In this section we want to present some code segments which colleagues where interested in as they want to reuse it in their work on other courses. We will give only an overview of every code part as we commented the code very extensive for this work.

A.1. Limit the movement

The following function is used to update the elements of the simulation. The code example is from the Point Charges simulation, hence the elements which are updated are point charges. This function exists similarly in the two other extended simulations.

With this function we limit the movement of the charge thats used for searching the hidden charges. As we described in our work this charge should only be movable between the inner and outer box. The simulation itself limits every movement to inside the outer box so we need to implement only the limitation to outside the inner box.

The box size of the outer box is 20 (unit less). The inner box is size of 16. The boxes are centered around the zero point of the environment. Hence the walls of the outer box are plus/minus 10 away from the zero point and the walls of the inner box plus/minus 8.

```
// !!! Sets the limits of the moveable region of elements !!!
// Updates the elements of the simulation.
// Is called while moving a charge.
// Takes the parameter boolean initial which is true when function
   called the first time.
private void updateScene(boolean initial)
{
```

```
double maxDist = 20.0;

// Initial placement of all charges
if (initial == true)
{
  engine.requestReorder( pc1 );
  setChargePosition( pc1, maxDist );
  setChargeAppearance(pc1);
  for ( int i = 0; i < 5; i++ )
  {
    engine.requestReorder( hiddenCharges[i] );
    setChargePosition( hiddenCharges[i], maxDist );
    setChargeAppearance( hiddenCharges[i] );
  }
}

// Movement of the visible charge that is moved by the student for
    searching the hidden charges.
// Restrict movement between inner and outer boxes.
// Box size outer box: +-10
// Box size inner box: +-8
// Movement is limited to inside the outer box from the simulation.
     So we only need to restrict the movement outside the inner box.
if ( pc1.shape.isSelected )
{
  Vector3d pos = pc1.getPosition();
  pos.add(mouse.getObjectTransform());

  // Restrict the movement in x direction
  if ( pos.x > 8 || pos.x < -8 )
  {
    pc1.setPosition(pos);
    engine.requestReorder( pc1 );
    setChargePosition( pc1, maxDist );
    setChargeAppearance(pc1);
  }
  // Restrict the movement in y direction
  if ( pos.y > 8 || pos.y < -8 )
  {
    pc1.setPosition(pos);
    engine.requestReorder( pc1 );
    setChargePosition( pc1, maxDist );
    setChargeAppearance(pc1);
  }
}

if ( mouse.getObjectTransform().length() > 0 )
hideDLIC();
mouse.clearObjectTransform();
```

```
}
```

A.2. Place elements on a grid

Placing an element on a grid means to only let them rest on a finite number of points inside the environment. From the programmers point of view, placing on a grid means rounding the position values depending on the grid size. In our case we wanted to have a grid size of one, so we could round the values to integers. This is done on the end of every move, when the mouse button is released. The charge then jumps to the next grid point. This is important as without a grid finding the exact position of a hidden charge is nearly impossible.

The first function rounds the position of the charge. But as it could be possible that a charge would jump on the place where another charge already rests after rounding the current position the new position is evaluated by the second function below.

```
// Every time a dummy is moved, on releasing the mouse button the dummy jumps to the next grid position.
// The new position of the dummy is evaluated if there is already a dummy placed.
// If the dummy is set on top of another dummy, it is moved to the right of the previous placed dummy.
// The new calculated position is then again evaluated because there could also be a dummy placed.
// This is done for all used dummies after a move.
private void roundPositions()
{
  Vector3d curPos = new Vector3d(); // The current position where the dummy should be placed
  Vector3d evPos = new Vector3d();  // The position after evaluation

  for ( int i=0; i < posDummies.length; i++ ) //Loop over all used positive dummies
  {
    curPos = posDummies[i].getPosition(); // Get current position where the dummy should be placed
    curPos.x = Math.round(curPos.x); // Place it on the grid in x direction
    curPos.y = Math.round(curPos.y); // Place it on the grid in y direction
    while(true)
    {
      evPos = evaluatePos(curPos,i,1);  // Evaluate the position
      if ( evPos == curPos ) // If the current position was free, stop evaluating
        break;
      curPos = evPos; // If the current position is taken place the dummy on the new calculated position and evaluate again
    }
    posDummies[i].setPosition(curPos); // Place the dummy on the empty position
  }
  for ( int i=0; i < negDummies.length; i++ ) //Loop over all used negative dummies
  {
    ...
    // Same code as for positive dummies
    ...
  }
  updateBoxes(); // Update positions in the HTML host file
}
```

This function evaluates a position if it is free or if there is already a charge placed. If the position is already taken the evaluated position is moved to plus one in x direction. This new position is then evaluated again.

```
// Evaluates a dummy position.
// Takes the position which has to be verified, the dummy id and the dummy charge.
// The dummy charge is importand as the dummys are always created and placed in the simulation. Unused dummies are hidden and their charge is set to zero. It is possible to place a used dummy on an unused hidden dummy!
// The charge of the dummy has no effect on the field! It is only for displaying the correct color of the dummy!
private Vector3d evaluatePos(Vector3d newPos, int dummyId, int charge)
{
  for (int i = 0; i<5; i++) // Loop over all positive dummies
  {
    if ( i == dummyId && charge > 0 )
      continue; // This case would be a comparsion with itself
    Vector3d posPos = posDummies[i].getPosition();
    if( newPos.x==posPos.x && newPos.y==posPos.y && posDummies[i].charge!=0 )
      newPos.x += 1; // Move charge one step to the right
  }

  for (int i = 0; i<5; i++) // Loop over all negative dummies
  {
    if ( i == dummyId && charge < 0 )
      continue; // This case would be a comparsion with itself
    Vector3d negPos = negDummies[i].getPosition();
    if( newPos.x==negPos.x && newPos.y==negPos.y && negDummies[i].
        charge!=0 )
      newPos.x += 1; // Move charge one step to the right
  }
  return newPos;
}
```

A.3. Place and remove dummies

In this section we want to show the code of placing and removing dummies. The important thing is that the dummies always exist and are always placed somewhere in the simulation. By placing we just set them visible and by removing we set them invisible. The reason for this implementation is, that starting the simulation needs a little longer to load but when running it is faster as it does not have to create the dummy charges any more.

The first code part we want to show is the initialization of the positive and negative dummy elements.

```
...
// Create new point charges elements for the dummies
posDummies = new PointCharge[5];
negDummies = new PointCharge[5];
int psn_count = 0; // Used for making dummies pickable for the mouse
ShapeNode[] psndpn = new ShapeNode[10];
for ( int i =0; i < posDummies.length; i++ )
{
  posDummies[i] = new PointCharge();
  posDummies[i].radius = 0.4;
  posDummies[i].setMass(1.0);
  posDummies[i].shape = new SphereNode( posDummies[i].radius, 20,
      positiveChargeAppearance );

  // At this point we just want to create the elements, not place
      them or make them visible!
  posDummies[i].shape.setVisible(false);
  bg.addChild( posDummies[i].shape ); //This adds the element to the
      simulations environment
  psndpn[psn_count] = posDummies[i].shape;
  psn_count++;
}
for ( int i =0; i < negDummies.length; i++ )
```

```
{
  ...
  // Same code as above for positive dummies
  ...
}
...
```

With the following function we place a dummy. That means we set a positive or negative dummy visible, set the charge and place it to the left or right border of the simulation.

```
private void addDummy(double charge)
{
  PointCharge dummyCharge;
  Point3d dummyPos = new Point3d();
  dummyPos.z = 0.0; // z is always zero

  if (charge > 0)
  {
    // If five or more positive dummies are placed do nothing
    if ( numPosDummies >= 5 )
      return;
    dummyCharge = posDummies[numPosDummies];
    dummyPos.x = 10; // Place the positive dummies on the right
        border of the simulation
    dummyPos.y = -2 + numPosDummies; // Place the dummies next to
        each other
    numPosDummies++;
  }
  else
  {
    // If five or more negative dummies are placed do nothing
    if ( numNegDummies >= 5 )
      return;
    dummyCharge = negDummies[numNegDummies];
    dummyPos.x = -10; // Place the negative dummies on the left
        border of the simulation
    dummyPos.y = -2 + numNegDummies; // Place the dummies next to
        each other
    numNegDummies++;
  }

  // Set the charge for correct dummy coloring
  dummyCharge.charge = charge;
  // Set dummy position
  dummyCharge.setPosition(new Vector3d(dummyPos));
  setChargeAppearance( dummyCharge );
  // Set dummy charge visible
  dummyCharge.shape.setVisible( true );
}
```

The next function deletes, or better hides, all dummies. It just go through all the dummies and set them invisible. The dummy counters are set to zero. Finally all invisible dummies are placed at the zero point of the simulation. This is of course not necessary but especially for debugging it is always helpful to have a cleaned up simulation. With this function we can also see why it is not possible to delete individual dummies. The reason is that the dummies have no ID or anything to identify theme selfs. So there is no system which would support the selection of one dummy. The reason is again to keep the code small and the simulation fast.

```
private void removeAllDummies()
{
  numPosDummies = 0; \\ Set positive dummy counter to zero
  numNegDummies = 0; \\ Set negative dummy counter to zero
  // Loop over all dummies. One loop for positive and negative
      dummies.
  for ( int i = 0; i < 5; i ++ )
  {
    posDummies[i].shape.setVisible(false); //Just set the dummy
        invisible
    negDummies[i].shape.setVisible(false); //Just set the dummy
        invisible
  }
  clearDummyPositions(); // This function will place the dummies to
      the zero point of the simulation.
}
```

Appendix B.

Used Software and Hardware

For our work we used the following software:

- VirtualBox 4.3.8
- Vagrant 1.5.1
- Python 2.7.6
- GWT 2.6.1
- Git 1.9.4
- Eclipse Kepler SR2
- Windows 7 x64

The following Hardware was used:

PC 1:

- CPU: Intel i7 4x 3,2GHz
- RAM: 8GB
- Graphics card: NVIDIA GT620

PC 2:

- CPU: Intel Core2Quad 4x 2,4 GHz
- RAM: 6GB
- Graphics card: NVIDIA GT610

Appendix C.

Setup of a local edX test server

For testing we had a test server set up on our local PC. The main reason was the possibility of testing our code easier and faster. Additionally we were able to modify some parts of the edX server for debugging. During our work we had two versions of the edX server running. While our code worked well on the newer server, we had to modify some files of the older server.

In this chapter we do not want to give a whole installation manual as they are available online. Instead we want to describe the important steps which are necessary for running our simulations.

C.1. Setup of the edX test server - Version 2013

For this server we were using the *mitxvm-edx-platform-02sep13a.box* file. The manual can be found online at [1]. This server is used till today allthough it is old because it needs less RAM and so could run on older machines too.

With this server had some problems with the *set_state* function. The reason was that within this version the jsinput method was also older and not so well tested. Referring to the comments of the jsinput code it was tested only with a small example application since there were simply no courses using it at the time it was written. The problem we had was that jsinput was trying to call our *set_state* function which was, compared to the *set_state* function of the example application, way bigger and therefor needed longer to respond. So jsinput ran into a timeout and reported an error saying *set_state* is not working.

For fixing this problem we had to modify these three files of the edX server:

/edx-platform/common/static/js/capa/src/jsinput.js

/staticfiles/js/capa/src/jsinput.js

/staticfiles/xmodule_js/common_static/js/capa/src/jsinput.js

These files are all placed on the edX virtual machine in */home/vagrant/edx_all/*.

The code line we changed:

$(document).ready(setTimeout(walkDOM, 300));

We changed the 300 to 3000.

Remark: Making this change locally on the test server the code wont run any more on an other edX server! We made this change just for debugging to see where the error is. We did not change our code to run on an unmodified edX server since this problem was solved with the newer edX versions. But as we described, on older PCs this version is preferred and that is the reason we gave this description here.

C.2. Setup of the edX test server - Version 2014

For the new version of the edX server we were using the *20140908-mitx-kifli-fullstack.box* file. The manual for this course can be found online at [2].

With this version we had a problem of the structure of our course. In this newer version the edX courses which are created by edX studio had another structure then our handwritten course. This resulted in errors including files as edX could not find them. This happened only locally on our test server.

For solving this problem we had to run the following lines in the console:

vagrant ssh – sudo disable_mongo_static

vagrant ssh – sudo restart web_loloadx

vagrant reload

For more details on this changes please see above linked manual.

Bibliography

[1] Quick Start to working with the edX Platform.

[2] Running edX Locally.

[3] Tacy Adam, Hanson Robert, Essington Jason, and Anne Tökke. *GWT in Action*. Manning Publications Co., 2 edition, 2013.

[4] Stephen Brown. Back to the future with moocs. *ICICTE 2013. Proceedings*, pages 237–246, 2013.

[5] Govinda Clayton and Gizelis Theodora-Ismene. Learning through Simulation or Simulated Learning? An Investigation into the Effectiveness of Simulations as a Teaching Tool in Higher Education.

[6] Danny Crooltall, David and Oxford, Rebecca and Saunders. Towards a reconceptualization of Simulation: From Representation to Reality. *Simulation/Games for Learning : The journal of Sagset*, 17(4):147–171, 1987.

[7] Yehudit Judy Dori and John Belcher. How Does Technology-Enabled Active Learning Affect Undergraduate Students' Understanding of Electromagnetism Concepts? *Journal of the Learning Sciences*, 14(2):243–279, 2005.

[8] EdX. *Building and Running an edX Course*. edX documentation, 2014.

[9] A. S. Gibbons, P. G. Fairweather, and T. A. Anderson. Simulation and Computer-Based Instruction: A Future View. In *Instructional Development Paradigms*, chapter 43, pages 769—-801. 1997.

[10] Henry Jenkins, Eric Klopfer, Kurt Squire, and Philip Tan. Entering the Education Arcade. *Computers in Entertainment (CIE)*, 1(1):8:1–8:11, October 2003.

[11] Cristina Ioana Muntean. Raising engagement in e-learning through gamification. In *Proc. 6th International Conference on Virtual Learning ICVL*, pages 323–329, 2011.

[12] Saif Rayyan and Chester Chu. Spring 2014 8.02 MITx Mid-Semester Survey. Technical report, Massachusetts Institute of Technology, 2014.

[13] Pim van de Pavoordt. Gamification of education. 2012.

[14] Dongsong Zhang and JayF. Nunamaker. Powering e-learning in the new millennium: An overview of e-learning and enabling technology. *Information Systems Frontiers*, 5(2):207–218, 2003.