Mathematics Game for Preschool Children Using Teachable Agents and HTML5

Björn Norrliden

Mentor:
Magnus Haake

Examiner:
Joakim Eriksson

September 24, 2012

Abstract

This master thesis work examines the theory behind teachable agents, and applies it to young children in a computer game environment, which is implemented using the HTML5 standard and JavaScript. To make a rigid system, interaction design for young children, together with what forms a basic understanding of mathematics is also studied.

Preface

This master thesis work has been done as a part of the Computer Science and Engineering program at Lund University, Faculty of Engineering (LTH). The work has been carried out together with the Educational Technology Group, at Lund University Cognitive Science (LUCS).

For all the help and support I got, and for all the time spent teaching me about new domains, I would like to thank my mentor, Magnus Haake, and the other persons in our team who I worked with: Agneta Gulz, Layla Husain and Lisa Wallin.

Contents

Chapter 1

Introduction

For several years, the teaching of mathematics in Swedish schools has been criticised. Universities have seen a decline in the proficiency of the newcomers and have to teach them topics that they ought to already know. (Sandström, 2007).

As for the mathematics education on any level, it has been pointed out that students with special needs should be given more attention and being taught with alternative methods (Skolinspektionen, 2009).

The solution to the above stated problems are probably a mix of many different methods and tools. This work aims to produce such a tool that can be used at an early level, even in the preschool environment.

Research that has been done on so called Teachable Agents (which will be described in detail later) shows that this is one interesting alternative method to teach people new subjects. Teachable Agents has been proved to increase knowledge and understanding of different subjects, and for students in varying age. However, the conditions under which the present project has been carried out has never been examined in detail before: This project will apply Teachable Agents to mathematics and children in the age of four to seven. In addition, the project will try to aim specifically for children who show signs of having troubles understanding the most basic level of mathematics.

The tool that will be developed in this project is a computer game. The game will build around the ideas of Teachable Agents. It will be developed for modern web browsers, using the newly released standard HTML5 together with JavaScript. The focus of this master thesis is thus placed in the intersection of many subjects: Cognitive science, interaction design and computer science. The project is carried out by a team of people with knowledge in those subjects, as well as in game design.

Chapter 2

Theories and Methods

2.1 Teaching Mathematics

When teaching mathematics to children in the intended target group with an age of four to seven, there are some things that have to be considered. It is also important to notice that the project aims at helping those students who seem to have trouble developing the earliest strategies for solving simple mathematical problems.

2.1.1 Number Sense

Studies in American schools have found that children who get the right help with developing their mathematical skills early on, in preschool, will perform better when they start to attend regular school (Sharon A. Griffin, 1994). It is important that children develop what is called *number sense*. They must understand the fundamentals of numbers; how different numbers relate to each other in magnitude (one set is bigger or smaller than another), that they can be added or subtracted and, importantly, obtaining good methods for doing so. This is knowledge which later on should become intuitive.

One of the most fundamental mental representations that children seem to have for numbers are the number line (Sharon A. Griffin, 1994). In practice, counting on the fingers is an example of visualizing the mental number line. A mental number line, visualized or not, helps to decide the cardinal number of a set (i.e. how many of something there is in the set) as well as performing addition or subtraction to a given set. When children have problems with mathematics at this level, it can be because they lack the ability to make up a number line representation.

By producing games, specially aimed at promoting such a representation, it was found that many children actually can be taught to adapt the number line thinking. Later, these children performed better in school (Sharon A. Griffin, 1994). These games were also designed to take advantage of what the children already knew about the world around them, building upon that knowledge.

The game developed in this project will be inspired by these results and the games that lead to them. Especially, promoting the number line representation will be an important goal.

2.2 Teachable Agents

The main scientific field that stands behind the game that is being developed in this project is the field of *teachable agents*. In short, a teachable agent (TA) is a computer based character that a player (the student) can teach by interacting with it. By teaching the TA, the student will also learn the current subject for itself.

To describe what a teachable agent is and how it works, some other terms will first be discussed briefly. Traditionally, in computer games, the social entities present in games have been divided into the two categories *avatars* and *agents*. An avatar is the character in the game that represents the player. It is typically controlled directly by mouse, keyboard or joystick input. A famous example is Super Mario. An agent on the other hand is a character controlled by the computer. Sometimes it is referred to as an *non-player character* (NPC). Both friends and foes in the game can be an agent.

There are also *hybrids* between avatars and agents (Chase *et al.*, 2009). In games such as The Sims, the player has control over several characters in a simulated home environment. Thus, the characters are avatars of the player. But the characters can also take initiative and interact with the environment without being controlled by the player – a feature of agents rather than avatars. A teachable agent can be seen as such a hybrid.

In a game deploying teachable agents, the player never explicitly controls the teachable agent, but instead the player interacts directly with the gaming environment, or through an avatar. However, the TA will learn or be influenced by the players examples or instructions, and then act in conformance with what it has learned. In this aspect, the TA is not only an agent, but also an avatar of the player.

Both agents and avatars provide some positive features, from a pedagogical perspective, that have shown to be apparent in the hybrid TA as well (Chase *et al.*, 2009).

2.2.1 Protégé Effect

The *protégé effect* bases on a combination of some agent and avatar features (Chase *et al.*, 2009). Together, these features explain why the student will see the teachable agent as their protégé. Having the TA being treated as a protégé means that the student will take responsibility for it, and that the student therefore will make effort towards learning (Chase *et al.*, 2009). The parts which together composes the protégé effect are described below.

Ego-protective buffer Since the TA is partly an avatar, it inherits the avatar property of having an *ego-protective buffer*. This means that when the student makes a mistake, it can share the responsibility between itself and the TA (Chase *et al.*, 2009). By doing so, negative feelings that the student might have bout itself, will instead more or less be directed towards the TA instead. This might decrease the amount of motivation that the student can lose at a failure.

It can also be the case that the blame for the failure resides in the teaching of the TA. It is then the fault of the student, but it is not the core knowledge of the subject that is the matter, but the teaching ability. The student will try to improve her or his teaching instead of blaming her- or himself for rendering a mistake. This statement is also part of the ego-protective buffer.

Another positive effect of the ego-protective buffer is that it invites the student to take risks that it might not take in an ordinary classroom context. Even though the outcome of an activity is considered a failure, the student will most likely has learned something along the way. By sharing the responsibility of the outcome with the TA, more such activities might take place while playing a teachable agent game.

Adopting a View of TA Intelligence as Incremental It is suggested that people tend to have either an entity theory or an incremental theory about their own intelligence (Dweck, 2000). An entity believer does not think that her or his intelligence can change, but that it is static. A person who thinks the opposite, that her or his intelligence can change is, on the other hand, said to have an incremental theory about her or his mind.

One of the components of the ego protective buffer is this theory (Chase *et al.*, 2009). Seeing one's own intelligence as static may be particularly true for low achieving students, as well as those with low self-confidence. It might be hard to change this fact, but an interesting finding is that those students might not have the same theory about *other* people. The fact that the teachable agent, like ordinary computer agents, are treated like real social beings, makes it possible to overcome an entity belief. By seeing that the TA actually learns, it should be obvious that the TA's intelligence fits with the incremental theory.

This fact is important for students with both entity and incremental beliefs about their intelligence, since it of course is important to understand that the TA can be taught anything at all. However, it may be even more important for the students with an entity belief about their intelligence. When interacting with the TA, such a student will not focus on incrementing its own intelligence, but the intelligence of the TA. In an ordinary scenario, the student might even give up too soon, thinking that it can not teach it self. Using a TA, however, the student will actually continue its studies to be able to teach the agent.

Sense of Responsibility It has been shown that people usually, unconsciously, view computers and other media devices as social actors (Byron Reeves, 1996). It is true both for the computer device itself, but also for digital characters, such as agents.

This has also been shown to be true for teachable agents (Dweck, 2000). As a consequence, just as with the pet toy called Tamagotchi, the player will feel responsible for the teachable agent. This will motivate the student to study more, which in turn will make it possible to teach the TA better. As a hidden consequence, the student will learn even more for her- or himself.

By allowing the player to pick among pre-designed TA's, or by letting the player customize one by itself, the player will get to interact with an agent that they hopefully like. The behaviour of the agent, i.e. the core principles of teachable agents, will of course not be allowed to tangle with, but other attributes such as gender or appearance can be. By allowing this, the player might feel even more responsible and connected to the teachable agent.

2.2.2 Offer New Ways of Thinking and Reasoning

An agent can function as a role-model for the player. This is common in traditional learning software, where the player simply learns from the computer, represented by the agent. Even though the concept of teachable agents can be seen as the opposite, since the player is teaching the TA, the goal is still that it is the player that should learn something. The positive feature from traditional agents (that the player can learn from them) is actually applied to TA's as well. However, to accomplish this, the TA must be implemented carefully. The TA can not behave like it knows something that the player has not yet taught it. Instead, the agent will visualize what it has learned from the player. When doing so, it can make its thinking visible. By explaining its decisions and choices in a good way, it can help the player to organize and structure its thoughts. Even though the player might know the answer to a certain question, it might not be sure about the exact reasoning behind it, or it might not possess a reasoning pattern that is as good as the one of the TA.

2.2.3 Promote Self-Regulation

It has been shown for other TA games, that learning results are better if the agent behaves like a *good student* and promotes self-regulation (Biswas *et al.*, 2005). This can be done by introducing spontaneity and initiative in the agent. The TA can reflect on feedback it gets from the player, react on misconceptions and ask for help if it runs into problems. One goal is that the student should think of what the TA is doing, and not just assuming that everything is fine. Even if the TA has been taught well, it can still do a mistake, and that should be noted by the student.

2.2.4 Possible Pitfalls

It important to note that most of the teachable agent theory is based on studies which did not include children such young as in this project's intended target group. Even though the studies are well done and proves that the theory works for older children, it cannot be assumed that it actually works for a four year old child. Possible problems will be stated here, as well as – when it seems possible – suggested solutions to them.

Understanding the TA as a Social Entity It has been shown that people in general recognize computers as well as characters presented by the computer as social entities (Byron Reeves, 1996). A digital character in a computer game will fuel psychological social responses in the user, just as if the user was being confronted by a real human. This fact is of course critical when it comes to understanding the TA as a social entity. However, it might be questionable if this fact holds for the target group. It is stated that even very simple representations, such as plain text, of a digital character will awake social responses (Byron Reeves, 1996), but for a very young child, more realism might be needed. However, children seem to be able to grasp characters as social entities when playing with toys or when watching television. To ensure that a TA is thought of as social, special attention needs to be put into the design of it. A simple representation is probably not enough, but good speech and animation modules should be added to the agent to minimize this possible problem.

Theory of Mind If a person has a *theory of mind*, it means that this person can attribute mental states to other persons (Premack & Woodruff, 1978). Such a state can include, among other categories, thought, beliefs, intentions or likings.

Naturally, most people are able to do this, and it paves the way to understanding that the teachable agent needs to be taught. The user understands that the teachable agent not necessarily knows what the user knows.

However, it is not clear that all the children in the target group can have a theory of mind, as this ability seems to be in development around the age of four (Wimmer & Perner, 1983).

Avoiding this hazard is not easy, and it might be a problem for the youngest children in the target group. Hopefully, the development of this ability can be catalysed by letting the children play the TA game.

2.3 Interaction Design for Children

The target group for the application being developed in this project is relatively wide when it comes to age, but this part of the project focuses mainly on 4-7 year old children. Much of the traditional human-computer interaction theories can not be applied because of this.

It can not be assumed that children it this age can read. Therefore, the amount of text in the game should be minimal, and – if used at all – complemented by some other form of coding, such as colours or voices (Ben Shneiderman, 2005) (Nam, 2010).

When it comes to buttons and other interactable objects, they should be few but large in size (Ben Shneiderman, 2005) (Nam, 2010). This is because the accuracy of small children is not enough developed to pinpoint small objects without effort. Also, the attention span might not be as high as for older users (Ben Shneiderman, 2005), making it important to not have more objects to interact with than is really needed. To focus the attention to these objects, is another argument to increase their sizes.

A study has shown that sometimes there are specific features in an application, which might gain from being designed somewhat different when a 3-5 year old child is supposed to use it (Nam, 2010). However, a redesign might still not be the best alternative. Consider the commonly used drag-and-drop function in many modern computer devices: First the user clicks on the object to be moved, and while holding the mouse button pressed, the object can then be dragged to the desired location, on which the object is placed by releasing the mouse button. For younger children, it was shown that it was easier for them to use another strategy: Click and release the mouse button once to pick up the object, drag the mouse without having the button pressed, and finally release the object by once again clicking and releasing the button. However, this operation might still not be the best alternative (that is, if the drag-and-drop functionality is to be used at all). This is because a child is often surrounded by older siblings, parents or tutors while they are utilising the computer. When the child runs into a problem, some one else will likely be called on for help, and then it is important that they can also interact with the system without too much confusion or effort. Because of that, it might be better to have the ordinary drag-and-drop system (Nam, 2010). The lesson from this example is that the user experience should not only take the children into account, but also the

people around them.

More importantly, having any drag-and-drop function at all – or having the need to move objects over the screen, should be avoided anyway, since it is hard for children ageing 3-5 to maneuver such a mechanism (Ben Shneiderman, 2005) (Nam, 2010). The same goes for double-clicking (Ben Shneiderman, 2005).

2.3.1 Education

Having a TA more or less implies the use of a computer software. But the decision to make a computer game to educate can be more motivated: A study has shown that both *multimodality* and *interactivity*, both of which are prominent features of computer games, increase the educational impact of the product they are part of (Ritterfeld *et al.*, 2009). Multimodality simply describes the state when there are many different channels in which the software can communicate with the user. By implementing visual, auditory, haptic and other output, the modality is of a higher degree then if just visual output had been used. The study showed that a higher degree of modality gives a positive result, even if the interactivity is static (Ritterfeld *et al.*, 2009). The interactivity, on the other hand, proves to be important as well. If the user is offered more and different opportunities to interact with the application, the educational outcome if better than if less interactivity has been used (Ritterfeld *et al.*, 2009).

Since both interactivity and multimodality are important features of computer games, it can be concluded that a computer game indeed can help to educate people. However, both of the factors must be implemented carefully. It is not certain that young children can handle too much information at the same time. If a high degree of modality is used, it should be to communicate *one* message at the time, to strengthen that particular message. When it comes to interactivity, it must be remembered that the child must be focused on the particular task in the game, which should strive to educate a given subject. Having to much interaction opportunities can possibly distract the child.

2.3.2 Feedback

To have a good feedback system in a software application is essential, since it will help the user understand what is going on and what kind of progress is being made (Norman, 2002). Feedback can be divided into different categories; one division is *information feedback*, *consequence feedback* and *point-based feedback* (McNamara *et al.*, 2009). They all have different benefits and drawbacks, but used in the right way, they should be applicable to the teachable agent game.

Information feedback is performed by simply informing the user what has happened, be it by a voice or text. In a teachable agent game, such a feedback could be given by the TA itself, by letting it talk to the child. Using text as information feedback should of course be avoided, since it can not be assumed that children in the target group can read. Since many children don't like to be patronized (Ben Shneiderman, 2005), it is important to design the information feedback in such a way that it is instead encouraging or positively helpful.

Consequence feedback is the kind of feedback given directly by changes in the environment, depending on the input. Instead of letting the system or an agent tell the

user what has happened, it can actually be seen what the consequences of the act were. Suppose that the child player is about to fill a bucket with a stated volume of water. If too much water is poured into the bucket, the water will spill over. It is then the spill over itself that is the consequence feedback.

Finally, the game could contain a point-based feedback system. This is simply the act of showing a score with a number or some kind of progress bar in the game. It shows how much progress that has been made.

For children, the easiest form of feedback to analyse is probably the consequence feedback, so that is what should be predominant in the teachable agent game.

It is interesting to note that the TA itself should be a good transmitter of feedback, due to its social properties. When interacting with social representations on a computer screen, users expect them to behave like real human beings when it comes to giving feedback (Byron Reeves, 1996). Information feedback should be easy to send from the agent, by simply letting it speak. But it is also important to give feedback showing that the agent is following along in what it is being taught. It should nod its head, make gestures or appropriate facial expressions, depending on what is going on in the game.

2.4 Developing Games With HTML5

2.4.1 Introduction

When a web page is downloaded to a user's web browser, it is encoded in some sort of format. Since the beginning of the World Wide Web, it has been the Hyper Text Markup Language (HTML). Early on, only simple content such as text, hyperlinks and images could be rendered. To show more advanced material, additional software had to be added, such as Adobe Flash. Such extensions are usually not available by default in web browsers, which has been a problem for developers who want the functionality of the web page to be present on many platforms (different operating systems, tablets, phones and so on).

2.4.2 The Canvas Element

Over the years, however, HTML has developed. The latest version is named HTML5 and it has elevated the functionality level of the language significantly. For this project, the most relevant addition is the canvas element, which allows software to draw arbitrary shapes in the web browser, and then control and animate them using JavaScript. This is ideal for games. Because of that, in addition with the emerging wide range support for HTML5 in phones, tablets and computers, this technology was chosen to implement the teachable agent game.

There are specified commands that can be send to the canvas element, to make it draw different shapes. By redrawing the canvas, animations can be achieved.

2.4.3 JavaScript

JavaScript is a scripting language. Scripts written in JavaScript can be downloaded together with an ordinary HTML web page. The script can then dynamically interact with

the web page that has been downloaded, and change its content. HTML is static, thus once a HTML web page has been downloaded to the web browser, it can not be changed without using tools like JavaScript. In short, JavaScript makes web pages dynamic. Therefore, JavaScript is essential if a game is to be developed for HTML5.

Syntactically, JavaScript look like conventional programming languages like Java or C++. However, it differs quite much. It is object oriented, but it is not class based like Java and C++. Instead, it is prototype based (Crockford, 2008). Instead of classes and objects, there are only objects. An object can have an internal link to another object – called the prototype. If an object has a prototype link to another object, it can use the prototype object's functions or object values. This is the construct that can be used to reuse code instead of classes.

To simplify and optimize the usage of the HTML5 canvas element, some JavaScript software libraries were used. Most prominent, a library called KineticJS (Rowell, 2012) was used. The library operates at a higher abstraction level, and makes interaction with the canvas element easier. For example, to draw a square on the canvas, several commands need to be sent to the canvas (at least one command for each corner of the square, then additional commands for styling). With the higher level library, there is instead a simple command for just adding a square. Other methods are included for facilitating the drawing of other shapes, including images, animating, styling and handling user interactions, such as mouse clicks.

It is also possible to program plug-ins to KineticJS. If a special graphical shape or other game entity will be needed a lot in the application, it might benefit from being written as a plug-in to KineticJS, as it will be easy to re-use this object at several places in the code.

2.4.4 Model-View-Controller

When developing the game, the design pattern Model-View-Controller (MVC) was used (see Figure 2.1). What it does is that it separate the core functionality of the application from the visual representation. The model part, which contains the logic of the application, will have no idea that it is presented on a computer screen. It just performs the logic operations. As an example, consider that the user clicks on a fish in a fishing game. The click will be passed to the model of the game, which realise that the user wants to catch the clicked fish.

Perhaps this will change the state of the game so that a fish entity disappears from the virtual pond, and is put in a basket instead. The fish has been caught in the model of the game. This is probably easy to implement, with just a few simple data structures and appropriate functions to alter these structures. To visualise this, however, quite different code might be needed, and it can even need a lot of graphics code. The MVC pattern untangles these two parts of the code. The View part of the application will observe the Model part. The Model part of the code should not "know" anything about the fact that the View is watching it. When the View part observes that a fish has been caught, it will start an animation catching a fish.

However, the Model need to broadcast a notification to its observing Views, when it thinks that something important has happened. The Model still don't has to know if there is any View observing it, it just broadcasts the notification, not caring if anybody gets

it. The View on the other hand, can choose which notifications it thinks are important, and just ignore the others.

The Controller part is intended to manage user input, and make sure that the Model and the View communicates in the right way.

There are several advantages of this pattern. Since the Model must be written in such a way that it does not know about the View that is observing it, one View can be replaced by another View. A 2D game can thus be updated to a 3D version, without needing to alter the game logic. Only the View has to be replaced. However, in this project that is not the main advantage.

Since the code for graphics and the model is separated, a graphical designer with basic coding knowledge can alter the visual appearance of the game, while being sure that the core functionality of the game is still intact, since it is located elsewhere.

Another good thing is, that there can be multiple Views at the same time, and that such a View does not need to be graphical. Such a view can for example be a logging device. Just as the graphical View, it observes the game Model, and writes events that it considers important to a logging file. Since this game is intended to be a tool used in research, what is interesting to log might change during time and with research interests. Using a MVC View as a logger, it should be easy to change what should be logged without having to modify the game logic itself.

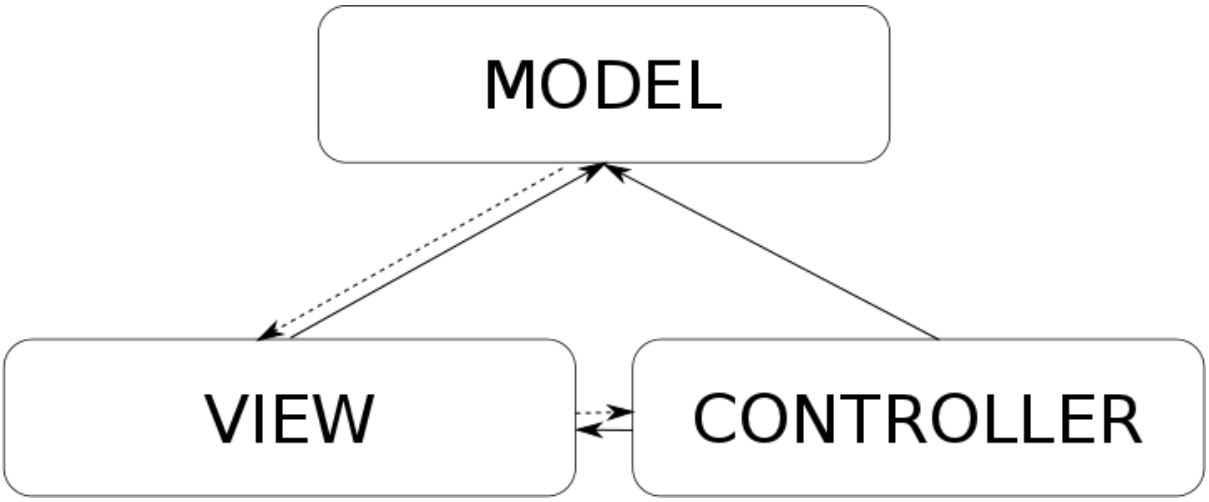

Figure 2.1: The Model-View-Controller (MVC) pattern illustrated. Dashed lines represent indirect communication between modules (could be through a message handler), while solid lines represent direct communication. The Model does not know anything about the View that is observing it, and can only communicate with its Views indirectly.

2.4.5 Compiling JavaScript

Since JavaScript is a scripting language, it is not compiled, but instead the source code is directly interpreted by a web browser. Still, there are some tools that can interpret the code, and then optimize it in different ways. The result will still be JavaScript code, unreadable for humans however.

Usually, compilation is considered to be a translation from language A to language B. For this project, the Google Closure Compiler was used to optimize the code, and then the translation will be from JavaScript to JavaScript. This is still considered a compilation, even though the target language is the same as the source language.

Apart from optimizing the code, which will make it run faster, the compiler can also reduce the size of the code, and place all of it in one single file. This will largely reduce the number of files that the web browser needs to download, as well as the total number of bytes.

2.5 Design Considerations

This section describes the main considerations and paths that the project took during development of the game, but that were eventually redesigned or taken away from the game for various reasons.

2.5.1 Minigames

To be able to include different teaching concepts in the game, the concept of *minigames* were introduced. The system with minigames were kept in the final prototype, but the way they were connected changed. Basically, a minigame is a short game included in the main game. The minigame teaches a specific mathematical concept, with some specific theory, using a teachable agent. The minigames connects with a framework story in which a teachable agent – a monkey – walks through a jungle to grow stronger to help its family who are in trouble. It was the framework story that later changed.

2.5.2 Fishing Game and Framework Story

The very first draft for a minigame was the *Fishing Game*. It was here that the basic structure for the teachable agent system was first seen: First, the child player plays the game by itself, then the agent watches the child play and finally the agent plays by itself. The number of mistakes the agent makes is about the same as the number of mistakes the child did while the agent was watching.

The fishing game itself consists of a pond with fish in it. The fish has different numbers on it, and the player is instructed to catch fish with a specific number. After enough correct fish have been caught, the player counts the number of fish with the correct number on it.

In the game window, an image of the child's avatar was showing while the child was playing. When the agent played, an image of the agent was shown instead.

However, after a working prototype of the Fishing Game had been developed, it was evaluated and eventually it was decided to drop that minigame. The main reason was that it did not include enough elements to promote number sense (described in Section 2.1.1). By simply catching a fish with a number on it, will just make the child recognise different numbers, i.e. how they are drawn or written. But the actual *meaning* of the number is not taught at all.

Another element that was decided to be removed from the game was the child avatar. The initial thought was that the child should be able to pick an avatar with an image

that they liked, to graphically represent the child in the game. It is however not obvious that a child in the desired target group would understand the concept of having an avatar representing themselves without to much effort from the young child's part. There was already questionable if the child would understand the concept of the TA, and another rather complex graphical and social entity in the game was considered unnecessary.

The idea of choosing an avatar of the child's liking was instead transferred to the teachable agent. Instead of always having a monkey, the child should be able to pick an agent who they liked, to higher the protégé effect. By doing so, the framework story with a jungle did not work, since the child should be able to pick an agent who don't live in the jungle. As stated above, the concept of having any sort of framework story connecting the minigames were however kept.

Also, the basic teachable agent system itself, were the child and the agent takes turns in playing, was kept during further development.

2.5.3 Guardian Angel

Initially, there was another character in the game, called the *guardian angel*. If the child player repeatedly failed to complete a certain level, the teachable agent would disappear and temporarily be replaced by a guardian angel. The guardian angel would then play the game correctly, and at the same time explain why its choices were correct. Here, the guardian angel took the role of a classical learning game agent, and it just tried to teach the player. This character was later removed, since too many social entities and playing contexts in the game could be confusing, and the function of the guardian angel could instead be moved to the teachable agent instead. This is further described in section 3.1.2.

Chapter 3

Result and Discussion

3.1 Game Concept

As stated in the design considerations (section 2.5), the game should consist of several minigames played in a series and connected by a framework story.

3.1.1 Watering a Garden

The new framework story included the teachable agent having a quest of collecting water drops. These water drops could later be used to water a garden belonging to the TA. The better results the child and the agent together renders, the more water drops the agent can collect. By showing the garden growing as the game goes on, a clear and intuitive indicator of the child's progress is illustrated. For the child however, the main function of the garden is to motivate further playing.

3.1.2 Playing Modes

In each minigame, the playing is done through three different *modes*:

Child Play In the Child Play mode, it is the child player who plays the minigame. The TA is only present as a small icon of it in the outskirts of the scene. The agent does not comment anything, and the player can not interact with the agent. This mode is intended to make the child comfortable with the specific minigame, before actually trying to teach the agent anything. This mode goes on for a specified number of rounds or time, before the system automatically switches to the next mode.

Agent See When the child play mode is over, Agent See mode is started. It is exactly the same as child play mode, but now the agent is present, and it comments on the player's progress. If the child player makes a mistake, the agent gives a leading comment on how it perhaps can be corrected. Still, it is the agent who should be taught, so it can not seem like the agent knows the correct answer, it can only give a rough suggestion. This is the function that replaced the idea of a guardian angel, as described in section 2.5.3. During this mode, the system records what the child player is doing, and the result of that will be saved and it will later influence how well the TA performs.

Agent Do The Agent Do mode is the last minigame mode. Now the agent takes over the controls, and the player will have to watch. Depending on previous achievements by the player and the agent in similar minigames, the agent will make more or less mistakes. One important feature in this mode, is that the child player has the opportunity to interrupt the agent when it is about to make a mistake. After interrupting, the controls are given to the child for one round, and the child can then correct the agent, providing the right answer. If the child misses an opportunity to correct the agent, or if the child interrupts the agent but fails to provide a correct answer itself, it will affect the learning curve of the TA negatively. The purpose of the interrupt button is to keep the child active, even though it is the agent who is playing the minigame. The child must still check what is going on, and take responsibility for the agent's learning.

It is the playing of the child in the Agent See mode and the interruptions of the agent in the Agent Do mode which renders a positive or negative impact on the TA's understanding of the current subject.

3.1.3 Ladder Metaphor

In the end of the project, the prototype of the game included two minigames, both of which build upon the *ladder metaphor*. The minigames promoted the concept of a number line (see Section 2.1.1) with different representations. The graphical representations were vertical, with bigger numbers located higher up – thus the name ladder metaphor.

Tree Minigame The Tree Minigame was the first to be developed after the Fishing minigame had been dropped, and the ladder metaphor introduced in the project. The game scene consists of a tree, located on the other side of a pit. The tree has ordered levels marked on it, ranging from 1 near the roots, to 6 near the tree crown. Inviting treats are placed in the tree, and the player is supposed to get the treats to the other side of the pit. By clicking on the correct number on a number pad, a creature will fly to or climb up the tree to get the treat for the player. When the agent is active, either by watching the child playing or playing itself, each treat will consist of a water drop for the garden.

Just as all other minigames are supposed to, the Tree Minigame follows the pattern of applying different playing modes. In Figure 3.1, the Tree Minigame is shown in the first mode – Child Play mode. The agent is only present as a small icon in the corner, and it makes no comments on the child's play. Also in Figure 3.1, the number pad has just been used, which is clearly shown by shading the pressed button differently from the others. A lizard has climbed the tree, and is just about to catch the parcel on the correct level.

In Figure 3.2, the Agent See mode has started. Now, the agent is more apparent in the scene, and makes comments on the playing. It is still the child who has the controls though. The bucket of water has been filled to about two thirds, which can be seen in the upper right corner in Figure 3.2 as well.

Finally, the agent takes over the controls in Agent Do mode. This is shown in Figure 3.3. The agent interacts directly with the number pad, which is shown by animating its arm to the button it is about to press. If the child player realises that the agent is about to make a mistake, the child has some time to react and press the sad-face-button.

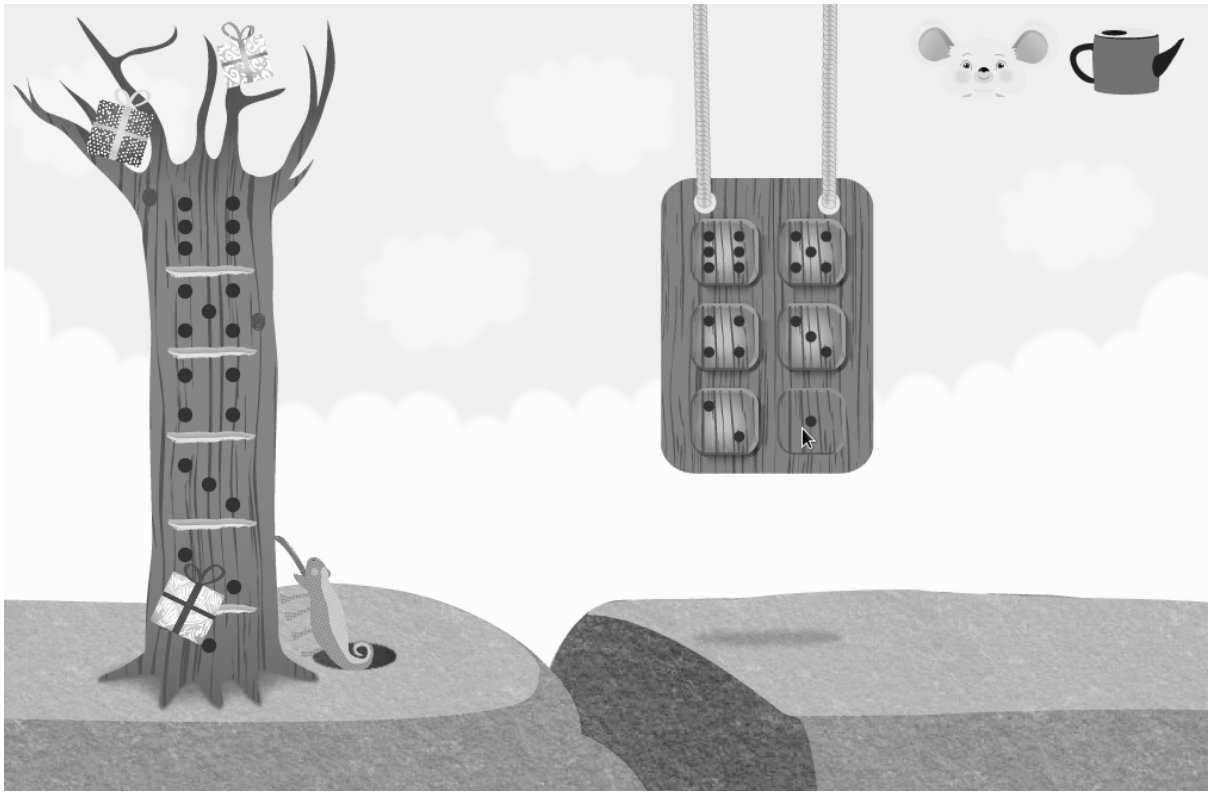

Figure 3.1: Tree Minigame. The player has just instructed its little helper to get the parcel on level one in the tree. The game mode is Child Play, which means that the agent will make no comments, and just reside silently in the upper right corner. The bucket in the same corner shows the progress of how many water drops that have been collected.

The controls are handed to the child for one try, before it once again goes back to the agent.

The game is implemented so that it is easy to replace the dots on the tree and the number pad with other number representations. The dots could instead be dashed, digits or fingers. One option is even to not have anything at all at the tree. The student must the learn the concept of *estimation* to approximately identify the correct level on the tree. The idea is that the same minigame can be played in different settings. When playing for the first time, the dots will be used. Later on – when the student (and the TA) seems to be able to play the minigame without problems, the dots can be replaced by digits on either the tree, on the number pad or on both of them.

Mountain Minigame Another version of the ladder metaphor was developed, and called the Mountain Minigame. The concept is exactly the same as in the Tree Minigame, but in this game, the player picks a correct number of balloons, to place on a cage. More balloons will make the cage fly higher. The goal is to get the cage to fly to the exact level on a mountain, were a friend in need is trapped.

The idea behind having almost the same game, but with a different visual representation of the ladder metaphor, builds upon the fact that the same subject should repeatedly be taught to the student. It is the same reasoning that replaces the dots in the tree game

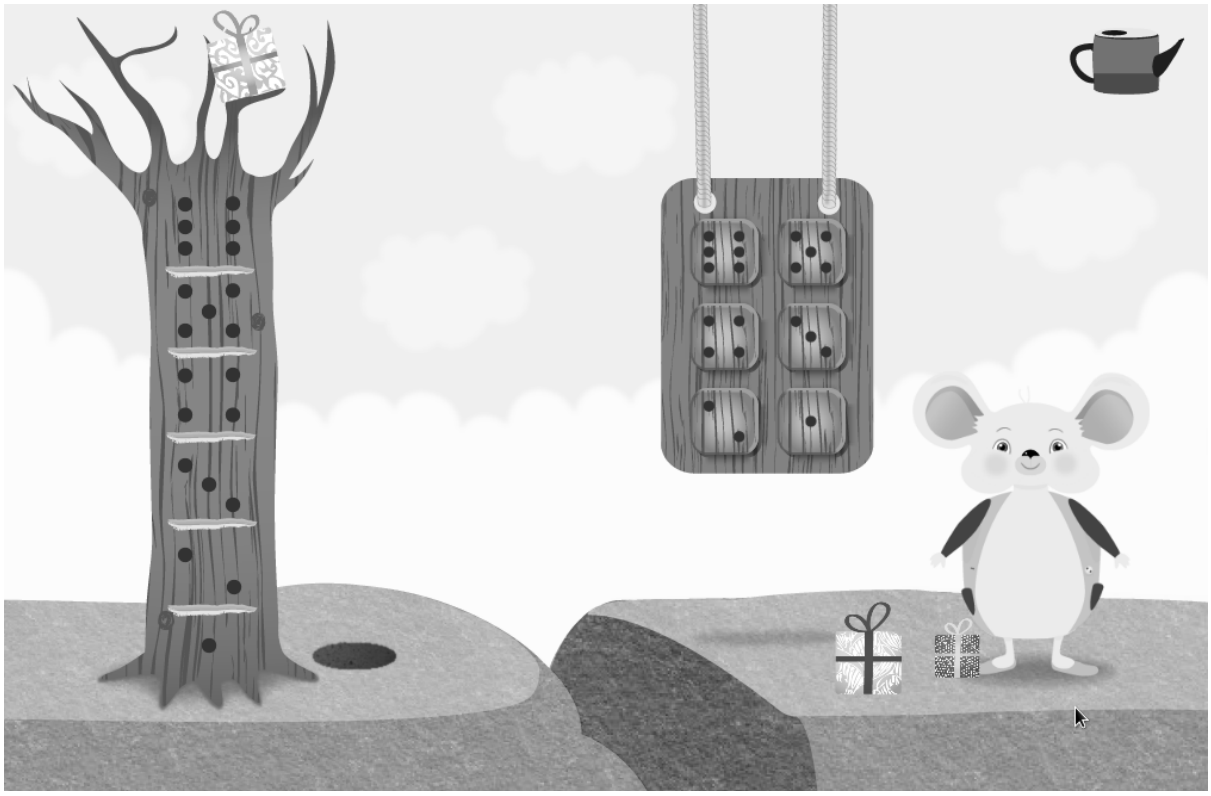

Figure 3.2: Tree Minigame in Agent See mode. The agent is now more present on the screen, but it is still the child who plays the game. During this phase, the agent learns what the child is doing.

with digits when the student reaches a higher level of understanding. To be able to repeat almost the same game concept, without the student getting tired of the games, more variation must be included. This is why both the Ladder Minigame and the Mountain Minigame was implemented. Ideally, more games that build upon the ladder metaphor should be implemented.

3.1.4 Feedback

The different forms of feedback discussed in Section 2.3.2 are present in the implementation. The most important one might be the consequence feedback, since it is easiest for a young child to recognize and understand. In the Tree minigame, that kind of feedback is what happens when the lizard in the tree climbs too low or too high. The consequence is that the parcel won't be acquired. In the Mountain minigame the same happens when the cage flies too high or too low, so that the friend in need can't be saved. There is really no need to have any other feedback to explain to the child if the picked number was correct or not. This is also the case in Child Play mode. In Agent See and Agent Do mode however, the agent gives additional information feedback. This will of course strengthen the impression that something went wrong or right, but its main purpose is to show that the agent is actually trying to learn something. It gives feedback to show if it understands or not. The comments it gives are designed to be helpful to the child,

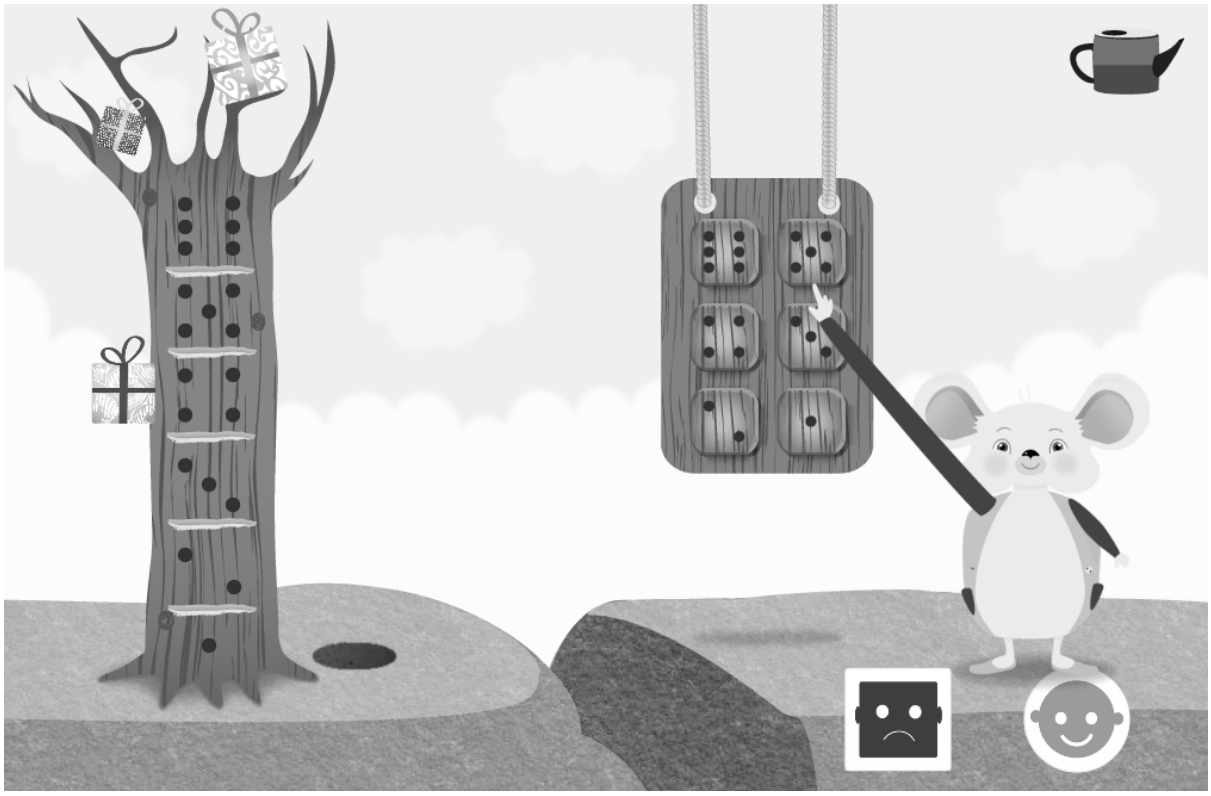

Figure 3.3: Tree Minigame in Agent Do mode. Now it is the agent who is playing. Its results are depending on what it learned in Agent See mode. The child player can interrupt the agent if it seems to do something wrong, by pressing the sad face button. It is sided by a button for encouraging the agent. When interrupting, the control temporarily goes to the child.

suggesting that for example more or less balloons should be attached to the cage. But that feedback can't be to detailed; the agent can only give vague suggestions, since it should not really know more about the area being taught than the child player does.

Facial expressions of the agent were also developed and discussed. Most basically, the agent can blink with its eyes, just to give a more living appearance and less robotic. But the important fact is that it can show a happy or confused facial expression, depending on what happens. When it is really happy, it makes a dance. This will hopefully strengthen the feedback from the agent, given in many cases. There are no sad or angry facial expression, since it is important to be encouraging to the child.

A point-based scoring system can be said to be implemented with the help of the garden. By showing the garden now and then, the progress of the playing is illustrated. However, this form of feedback might just as well be called consequential feedback, since the actions in the game directly affects the state of the garden.

3.2 Software

3.2.1 HTML5

Comparison to other tools

Using HTML5 proved to work well for developing computer games. Traditionally, the most dominant tool to develop games for web browsers has been the Flash technology, and HTML5 together with its Canvas element is often compared to Flash. One of the biggest advantages of using HTML5 is that it is already included in most modern web browsers, and a game can thus be started right away after browsing to its web site. While using Flash, a plug-in component must first be downloaded. A disadvantage of both technologies is the support on mobile devices. Both Flash and HTML5 is supported on some devices, but not all of them. The interesting fact about this is however that HTML5 will most likely *be* supported in the near future.

One of the advantages of Flash on the other hand, is the availability of graphical tools for the developer. To construct a scene, graphical tools exists, were the developer can drag-and-drop objects to a desired location, and easily animate them using timelines and other tools. When it comes to programming a HTML5 application, almost everything has to be programmed by the developer, and placing an object in its right place can be a time-consuming task of trial-and-error. On the other hand, many third party libraries are emerging to help the programmer (KineticJS being one). Timelines and animations are fairly easy to implement using such tools, even though no graphical interfaces are present.

Performance

To use HTML5 with good performance, hardware acceleration should be turned on by the web browser. This means that the browser uses the circuits on the computer that are dedicated to render graphics, thus increasing performance. Especially for mobile devices, this boost is not fully supported yet, but will likely be better in the future. When testing the teachable agent game on a modern desktop computer, it runs with a rate of at least 60 frames per second (FPS) – well enough to make it seem smooth and not rendering any annoyance. On a fairly modern mobile device, the measure only reaches 10 FPS, making it hardly playable. On a tablet device however, it runs well enough. Since one of the initial goals of the application was that it should be able to run on tablets, that is a good fact, even though improvement on the Canvas rendering would be welcome. As stated above, it can be expected.

3.2.2 Model-View-Controller

The design pattern Model-View-Controller (MVC) which was decided to use for the application, proved to be a good choice.

Event Handler

A special JavaScript event handler were written for the application, to which Model objects can send notifications when something important happens. View objects on the

other hand can subscribe to the event handler, and will get notified when something happens. It is common for MVC applications that specific View objects subscribes directly to specific Model objects, not having a message or event handler in between. Another common design is that the Model just send one type of notifications, leaving it to the View to realize what has happened.

For this application, however, an event handler was implemented, providing more flexibility between different Model and View objects. Instead of letting a View observe a specific Model, it observes specific event types, which any Model can send out. During further application development, it proved to work good and to be useful, but it also made it easier for the programmer to accidentally let the View objects receive notifications which should not be received. To avoid that problem, the event objects that were sent, had to be carefully assigned different names, so the programmer won't mistake one event type for another.

Animations and Callback functions

The MVC pattern needed some pattern breaking modifications as well. One of the biggest considerations were what should happen when a View needs to do an animation. As described in Section 2.4.4, the Model will send an notification when something it thinks is important has happened. As an example, when a fish has been caught in a fishing game, it might send a notification that a fish has been caught. The View will receive the notification and initiate an animation, showing a fish being caught from the pond and thrown to the fishing basket. In the Model, the fish had already been caught when the notification were send, and the Model is thus ready to go on with the game. But the View needs to finish its animation of this event first, and the Model thus have to wait for the View. This interferes with the MVC rule that the Model should operate independently from the Views observing it. However, it was decided for this application to add extra functionality to the event handler: When broadcasting a notification, the Model can chose to send an extra argument, saying that it will not continue executing until an attached callback function has been called by a View object. When the View has finished its animation, it will simply call this attached callback function. In this way, the Model still don't know anything about the Views observing it, even though it is dependant on it.

Replacing the Controller

Another adaption to the conditions were that the graphics library, KineticJS, were able to handle user input in a good way, and connecting it to graphical objects on the screen. Since the game only has mouse or touch input, it proved to be extremely helpful, and no other JavaScript functionality but the one from the graphics library were needed to detect such events. This made it natural to let the View objects handle user input, even though a dedicated Controller object should take care of this. However, using a Controller would most likely just have made the source code more messy than it needed to be, and since the input methods were so easy to handle, this never proved to be any problem.

The decision to allow callback functions in the MVC events can be seen as a compromise between strict following a programming pattern and actually using it. The same goes for the fact that the View objects handle user input. It is possible that another

design pattern would fit the game better, or that the current implementation should not be called MVC. However, the implementation proves to work good, and thus the name of the used pattern might not be that important.

Design Process Benefits of MVC

During the design phase of the game and its minigames, a simple graphical representation was used. It consisted of simple public domain images and basic geometrical shapes created using KineticJS. Later on, when the game mechanics of especially the Tree minigame were viewed as stable and good, the graphics began to be replaced by professional graphics, dedicated for this project only.

This is a very good case to see the benefits of the MVC pattern. Replacing the graphics was a rather big project, and a lot of code had to change or be rewritten. However, since the Model and the View were separated, changes only needed to be done in the View part. The Model, which was unit tested and declared stable, did not have to change at all in the process. This ensured that even though the graphics changed, the game would still continue to work as expected.

Another example of the strength of MVC is the fact that both the Tree minigame and the Mountain minigame build upon the ladder metaphor (see Section 3.1.3). This made it possible to implement *one* Model for both minigames – the Ladder Model. This model could be tested and kept very simple, excluding all the graphics. When the two different minigames run, it is only a different View that is pasted upon the Model, either the Tree View or the Mountain View. Solutions like this simplifies the development process a lot, and it also makes it easier to ensure that game models are working, since less code has to be tested or verified.

3.2.3 JavaScript objects and inheritance

Many object oriented programming languages, like Java or C++, objects are built from classes. The class can be seen as a template for the object. In JavaScript, classes do not exist, there are only objects. In this application, the scripting language were used in such a way that classes were simulated (Crockford, n.d.)(Resig, 2008). Inheritance is present in JavaScript, but treated in a different way, since classes are not present. But by simulating classes, "ordinary" inheritance can be achieved as well. This simulation take use of the prototypal nature of the language, but when using the classical inheritance mechanism, the prototype structures are hidden to the programmer. It is possible to write a large JavaScript application without simulating classes, and instead using true prototypal inheritance, but for this application the simulation of classes were viewed as something positive. It is known that in the future, other developers will take over the code base of the project, and they will most likely be trained in the class based approach.

3.2.4 JavaScript compilation

As stated in Section 2.4.5, a compilation tool (Google Closure Compiler) was used. Apart from optimizing the code and making it faster to download and run, it also provided some other functionalities that were good for application development. By allowing the

programmer to add special tags in the code, it could detect many mistakes, that the web browser interpreter could not. For example, with the Closure Compiler, it is possible to use strict typing (something that JavaScript don't have). Thus, an assignment of an integer to an KineticJS object variable will be reported as an error, even though the JavaScript language actually permits it.

Chapter 4

Conclusion and Further Work

The master thesis project has been carried out in the intersection of several subjects, the main ones being:

1. Explore the concept of teachable agents and how it can be applied to young children and basic mathematics.
2. Interaction design for young children.
3. Implementation of the above using HTML5.

The available theory on teachable agents (TA) is comprehensive, and helped to draw the initial guidelines for the project. However, it was soon found out that most of the theory is building on studies which has been carried out on grown-ups, or children older than in the present project's target group. Some potential pitfalls were pointed out, and special care were taken to avoid them. It is not evident that children can understand a TA as a social entity – something that the theory behing TA's build upon. The same argument goes for the theory of mind, which starts to develop in the age of the intended target group.

Unfortunately, no study has yet been carried out with the software developed, even though it will be in the future. This is however outside the scope of this master thesis.

When it came to teaching basic mathematics, number sense was discovered to be one of the most important subjects to focus on for such young children. Thus, the resulting computer game promotes number sense.

When designing interfaces for young children, special care has to be taken. It is interesting to note, however, that it is just as important to take into consideration that parents and teachers must be able to use the software as well. In the future, the software must be tested on the target group, to really ensure that the design decisions work as intended.

The implementation of the game was made using HTML5 and JavaScript. HTML5 is a relatively new standard, and not often used for making computer games (yet). It proved to work good, even though modern hardware and software must be in place for it to work. It is very likely that such prerequisites will become more and more available.

JavaScript works quite differently from classical object oriented languages. Even though it is not needed, the classical approach were simulated for this master thesis. It rendered a positive result.

References

Ben Shneiderman, Catherine Plaisant. 2005. *Designing the user interface: Strategies for effective human-computer interaction.* 4 edn. Pearson Education.

Biswas, Gautam, Schwartz, Daniel, Leelawong, Krittaya, & Vye, Nancy. 2005. Learning by teaching: A new agent paradigm for educational software. *Applied artificial intelligence*, **19**(March), 363–392.

Byron Reeves, Clifford Nass. 1996. *The media equation: How people treat computers, television, and new media like real people and places.* CSLI Publications.

Chase, Catherine C, Chin, Doris B, Oppezzo, Marily A, & Schwartz, Daniel L. 2009. Teachable agents and the protege effect: Increasing effort towards learning. *Journal of science education and technology*, **18**(4), 334–352.

Crockford, Douglas. Classical Inheritance in JavaScript. *http://javascript.crockford.com/inheritance.html.*

Crockford, Douglas. 2008. Prototypal Inheritance in JavaScript. *http://javascript.crockford.com/prototypal.html.*

Dweck, C.S. 2000. *Self-theories: Their role in motivation, personality, and development.* Essays in Social Psychology. Psychology Press.

McNamara, Danielle S., Jackson, G. Tanner, & Graesser, Arthur. 2009. *Intelligent Tutoring and Games (ITaG).* Los Angeles: University of Southern California.

Nam, Heather. 2010. Designing User Experiences for Children. *Uxmatters.*

Norman, Don. 2002. *The Design of Everyday Things.* Basic Books.

Premack, D., & Woodruff, G. 1978. Does the chimpanzee have a theory of mind? *Behavioral and brain sciences*, **1**(4), 515–526.

Resig, John. 2008. Simple JavaScript Inheritance. *http://ejohn.org/blog/simple-javascript-inheritance/.*

Ritterfeld, Ute, Shen, Cuihua, Wang, Hua, Nocera, Luciano, & Wong, Wee Ling. 2009. Multimodality and Interactivity: Connecting Properties of Serious Games with Educational Outcomes. *Cyberpsychology and Behavior*, **12**(6).

Rowell, Eric D. 2012. KineticJS. *http://www.kineticjs.com.*

Sandström, Anders. 2007. LTH sänker kraven på kurser i matematik. *Sydsvenskan*, October 20.

Sharon A. Griffin, Robbie Case, Robert S. Siegler. 1994. Rightstart: Providing the central conceptual prerequisites for first formal learning of arithmetic to students at risk for school failure. *Pages 25–49 of: Classroom lessons: Integrating cognitive theory and classroom practice*. Cambridge, MA, US: The MIT Press.

Skolinspektionen, Swedish Schools Inspectorate. 2009. *Undervisningen i matematik – utbildningens innehåll och ändamålsenlighet, kvalitetsgranskning rapport 2009:5*.

Wimmer, H, & Perner, J. 1983. Beliefs about Beliefs: Representation and the Containing Function of Wrong Beliefs in Young Children's Understanding of Deception. *Cognition*, **13**(1), 103–128.