

VRIJE UNIVERSITEIT AMSTERDAM

Amsterdam, The Netherlands

SERIOUS GAMES
IN
INTERACTIVE MEDIA

A thesis submitted in partial fulfillment of the
requirements for the degree of
MASTER OF COMPUTER SCIENCE

in

MULTIMEDIA

by

Celestyna Banaszak

November 2011

Table of Contents

1. Introduction.....	4
2. Role of Interactive Media in Education.....	6
2.1 Interactive Media.....	6
2.2 Examples of Learning Through Interactive Media Applications.....	8
2.3 Learning Paradigms for Game-Based Education.....	10
2.4 Serious Games as a Potential Cure for Poor Education.....	12
2.5 Designing a Serious Game.....	15
3. Technical background.....	16
3.1 XIMPEL.....	16
3.2 Adobe Flex, MXML, Action Script 3.0.....	19
4. Minigames.....	20
4.1 Introducing Minigames in XIMPEL.....	20
4.2 Minigame Templates.....	21
4.2.1 Memory Game.....	21
4.2.2 Matching Game.....	22
4.2.3 Sorting Game.....	24
4.2.4 Questionnaire.....	25
5. <i>Meet Rembrandt</i> with Minigames in XIMPEL.....	26
5.1 Part One: Getting To Know Rembrandt.....	26
5.2 Part Two: Rembrandt's Triumph.....	29
5.3 Part Three: The Twilight.....	33
6. Designing a Serious Game. Three versions of <i>Meet Rembrandt</i>	38
6.1 Branching Structure.....	38
6.2 Minigames.....	39
6.3 Navigation.....	40
6.4 User Interaction.....	41

6.5 The Effect of Minigames on the Storygraph.....	42
7.Feedback.....	44
8. Conclusions and Future Work.....	47
References.....	50
<i>Meet Rembrandt</i> - Resources.....	51
Appendix A.....	52
1. The Code of Minigame Media Type - minigame.mxml.....	52
2. Templates.....	53
2.1 Memory Game.....	53
2.2 Sorting Game.....	58
2.3 Matching Game.....	63
2.4 Questionnaire.....	69
2.4.1 questionnaire.mxml.....	69
2.4.2 Question.as.....	72
2.4.3 AnswerGroup.mxml.....	72
2.4.4 Answer.as.....	73
2.4.5 AnswerEvent.as.....	74
Appendix B. Tutorial: MINIGAMES in XIMPEL.....	75
1. Creating a Minigame.....	75
1.1 Making Minigames Compatible with XIMPEL Player.....	75
1.2 How Does It Work?.....	76
2. Templates.....	76
2.1 Memory Game.....	77
2.2 Matching Game.....	78
2.3 Sorting Game.....	80
2.4 Questionnaire.....	81
3. Adding a Minigame to the Playlist.....	82

1. Introduction

XIMPEL, the eXtensible Interactive Media Player for Entertainment and Learning, is a platform that offers the possibility to create interactive media applications. An intended purpose of XIMPEL is to design simple educational games. Currently, such games can be created by including video fragments and providing a navigation structure between them. For educational purposes this might not be sufficient. Marilyn Fenichel and Heidi A. Schweingruber showed that knowledge acquisition benefits from active engagement of the students [1]. To achieve such engagement in XIMPEL, I explore the use of minigames. Minigames are simple assignments that can be used to challenge the player, check their knowledge and help to memorize important content. They might also serve as a stimulating experience, from which students can learn.

To successfully integrate minigames into XIMPEL two issues need to be addressed. First, for authors it should be simple to include minigames in their application. The platform's modular approach allows for incorporation of user's own Flash games, however, currently, this process requires programming skills in Flex. The advantage of XIMPEL is that the applications can be easily created by modifying configuration and playlist files in XML. Thus, adding video clips and images to the playlist does not require programming knowledge. This should be also the case with minigames.

Second, the minigames should help users in the learning process. It is not clear how they could be used to make learning more efficient. Also, extending the platform with the possibility to add minigames to the playlist might influence the design of XIMPEL applications.

In my research I investigate:

- how to implement minigames in XIMPEL in such a way, that the user could easily employ them in their applications,
- how minigames can be used in interactive video,
- how such extensions would affect the design of XIMPEL applications in terms of a storygraph,
- how minigames could support learning.

To answer my first research question, I create an extension to the platform that allows minigames to be added to the playlist. In addition, I introduce templates for several minigames, that authors can use and configure

for their purpose. To illustrate how minigames can be used in interactive video, I present *Meet Rembrandt*, a game I created in XIMPEL. Comparing three parts of the game, I explore what the consequences are on the storygraphs, when minigames are included. I discuss how minigames may encourage players to study and help them acquire knowledge.

The structure of this thesis is as follows. In the second chapter I introduce the conceptual background knowledge, focusing on interactivity in the context of new media. I explain how this idea has developed in past years and what role it plays in computer- and video-based education. I present studies and examples proving the potential of interactive learning, and discuss scientific findings that support my view. I give hints on how to design a good serious game.

The third chapter concentrates on the technical background, explaining technologies used to create *Meet Rembrandt*. There, the reader can find an overview of features and possibilities of the interactive video platform XIMPEL, as well as a short presentation of required programming languages and frameworks: Adobe Flex, ActionScript and MXML.

In the fourth chapter I explain the process of introducing minigames in XIMPEL and present four game templates. I discuss their role in the playlist and how they aid learning.

In the next two chapters I introduce my showcase, an educational game about Rembrandt, created in XIMPEL, and extended with minigames. In chapter 5, I present the storygraph of *Meet Rembrandt*. In chapter 6, I reflect on my design considerations. Presenting three versions of my game, I show possible approaches (in terms of different branching structure, the balance between minigames and storytelling, navigation and user interaction) and discuss their advantages and drawbacks. I investigate what is the effect of minigames on the design of XIMPEL applications in terms of a storygraph and how they support learning.

In the end I present the results of my informal interviews that I have conducted in order to get feedback on my work. I mention issues for future work, that are out of the scope of this thesis, and draw conclusions from my research.

2. Role of Interactive Media in Education

The world needs well educated people, that not only got their degree, but also have gained knowledge, skills and experience. Unfortunately the academic education is not always considered appealing by the students. The studies done few years ago [3] showed that half of the engineering students have resigned. It happened not because of lack in their abilities (they had the same results as those who stayed), but because of poor teaching methods. What is even more interesting - these remarks were related to the form of the lectures, not the quality of teachers. To get involved, students need at least some kind of "interactive engagement". It helps them to stay focused on the content, encourages testing different ideas and motivates for learning.

In this chapter we explore how multimedia applications and, in particular, games can be used to achieve interactive engagement. I will first discuss the complex concept of interactivity in the context of new media and show examples of learning through interactive media applications. Then, I will present learning theories supporting the view, that game-based education may enhance learning process. In the end I will discuss the reasons why serious games may have the potential to cure poor education and give hints on how to design a good serious game.

2.1 Interactive Media

The concept of interactivity has become a widely recognized subject of exploration in the context of new media. Thus, it is important to have a basic understanding of new media before examining interactivity in more depth.

New media have been growing out of traditional media for some time. Until the 1980s media relied primarily upon print and analog broadcasts of television and radio. In the last 30 years they have transformed into digital technologies, such as the Internet or computer games. Other forms of new media include such technologies as: video telephones, electronic bulletin board systems, videotext, and teletext and other forms of interactive television [10].

Interactivity is generally considered to be a central characteristic of new media, but means different things to different people. Many scholars have observed that the term 'interactivity' is often undefined [10], so they have

begun to seek definitions by examining various characteristics of the new media environment. Some of them have suggested that it is important to make a distinction between different ways of interacting with new media. For example, Szuprowicz identified three levels of interactivity: user-to-user, user-to-documents, and user-to-computer (user-to-system). This three-dimensional construct seems to encompass the primary literature on interactivity in new media and parallels historical developments in the concept of interactivity that pre-dated new media. It has been evolving for decades and provides a basic framework for investigation of the past, present, and future of interactivity.

User-to-user interaction focuses on ways that people interact with each other. It is based on human communication research and predates new media, however, has been strongly influenced by them. Media such as computer networks and telecommunication systems add a layer of technology between communicating partners [10]. Among the new media that enable social uses are: email, networked electronic bulletin boards, chat, or electronic shopping. New media also introduce new tools enabling people to have more control over their communication experience and not being bound by constraints of time or geography.

User-to-documents interactivity occurs when people interact with documents and their creators. Active navigation of Web sites and active participation in creation of interactive fiction can be examples of such interactivity. The 'audience' is not only a passive receiver of information, but rather an active co-creator, who has control over both presentation and content [10].

The interaction between people and the computer, or other type of new media system, is called user-to-system interactivity. Typically, research in this field defines the interaction between a single human and a single computer as the most elemental form of interactivity [10]. The main question here is who is in control, the computer or the human interacting with it? The former means that the computer presents information to users, who are aware of this fact and respond by, for example, filling in Web-based forms. The latter assumes a more active user, who manipulates the computer in order to obtain information, using the interface provided by designers. It includes, for example, databases, spreadsheets and word processors, that are used by individuals to manipulate and organize data. When the computer is in command of the interaction, but is responsive to individual needs, we can talk about adaptive communication. This is used in advanced gaming and educational systems, which are able to adapt to changes in the users' skill level.

This classification helps to understand the complex term of interactivity, however should not be considered as mutually exclusive. Some areas among these traditions may overlap and some forms of interactivity and new media may not fit into any of these categories at all [10].

2.2 Examples of Learning Through Interactive Media Applications

Interactive media can play an important role in education, providing a rich and varied set of resources for learning, available to people in their daily lives. Although television is still the most widely used source of information, the Web starts to reach larger and more varied audiences [1]. Information has become broadly available through online resources and communities. Interactive media allow users to select appropriate material and often make it more accessible by giving a possibility to choose the language of a narrative or adjust other features. They make it easy for students to revisit specific parts of the environments to explore them more fully, to test ideas, and to receive feedback.

There are many different kinds of interactive experiences, that can spark interest and maintain learners' engagement while also increasing knowledge and provoking to think. Some of them are introduced in museums and involve touching, turning knobs, pushing buttons or doing other manipulations in order to move some objects or check answers. One such exhibit was designed to help visitors understand the form and function of the human skeleton [1]. Participants pedaled a stationary bicycle and could observe an image of a moving skeleton on a large pane of glass. According to museum researcher Jack Guichard, the exhibit experience seemed to transform children's understanding of the skeleton. After visiting the museum, 93 children aged 6-7 were given an outline of the human body and ask to draw a skeleton. The study shows that 96 percent were able to do it correctly, in contrast to a group of children that have not experienced the exhibit and where only 3 percent could draw a skeleton. The knowledge persisted over time, 92 percent of children who have attended the exhibit, still remembered the idea 8 months after their museum visit.

Other types of interactive experiences include, for example, those that occur in media. In the case of television, people may watch others participating in interactive experiences, or, in the case of new media, software responds to their actions. User-to-user interaction can be observed in

electronic mail or chats, that help students to communicate without being limited to face-to-face interaction. Video games played on a computer can serve as an example of user-to-system interaction. Their potential to enhance learning process will be discussed in more details in chapter 2.4.

The user-to-documents interactivity can be widely observed in the Internet, as well as in television. One of the earliest attempts to use video technology to introduce students to real-life problems was *The Voyage of the Mimi* [2]. It was a thirteen-episode American educational television program depicting the crew of the *Mimi* exploring the ocean and taking a census of humpback whales. The series was created by the Bank Street College of Education in 1984 to teach young people about science and mathematics in an interesting and interactive way. Each episode consisted of two fifteen minutes segments: the actual episode, followed by an "expedition". Each episode was an entertaining movie and a corresponding "expedition" was a documentary, that taught viewers some scientific issues related to plot events in the previous episode of the show. As an example, one of the episodes was about obtaining drinkable water, and over the course of the episode, the viewer would also be given lessons about condensation, heat and the three states of matter. Each lesson had accompanying student and teacher handouts and worksheets [15].

More recent examples include *The Adventures of Jasper Woodbury* [2], which consist of 12 interactive video environments that focus on mathematical problem finding and problem solving. They were designed for students in grades 5 and up. Each videodisc contains a short video adventure that ends in a complex challenge and can be revisited on a "just-in-time" basis as students need them to solve the Jasper challenges. Each adventure is designed from the perspective of the standards recommended by the National Council of Teachers of Mathematics. Students who worked with the series have shown progress in mathematical problem solving, communication abilities and attitudes towards mathematics.

Next to television shows, also computer-based interactive experiences have been introduced in the past. *The Little Planet Literacy Series* [2] helps first- and second-grade students improve their reading comprehension and ability to write. In one of the challenges, they need to write a book in order to save the creatures of the Little Planet. The series uses animated video stories, combined with computer software and proven instructional techniques and was introduced in about 1000 classrooms nationwide [12]. This example shows, that interactive video programs may contribute to learning in different fields, not only science.

The interactive video materials have positive effects on learning, but they should be used wisely. Research conducted at the Exploratorium in San Francisco shows that more interactive features are not necessarily better [1]. In an exhibit called *Glowing Worms*, museum developers introduced three different versions of the same experience, each with a different level of interactivity (high, lower, non-interactive). The interactive exhibits attracted visitors for a longer time, were more enjoyable and helped participants to reconstruct relevant details better than the non-interactive one. However, no difference was observed between the more and the less interactive exhibits. This shows, that adding more features does not necessarily enhance the experience. The researchers from the Exploratorium in San Francisco think that too many interactive features can lead to misunderstandings or cause visitors to feel overwhelmed, and that there may be an optimal degree of interactivity, which results in a satisfying learning experience for the majority of participants.

2.3 Learning Paradigms for Game-Based Education

There exist learning paradigms, that can apply to game-based education. Merrilea J. Mayo [3] summarized six observations to improve game-based education: experiential learning, inquiry-based learning, self-efficacy, goal setting, continuous feedback with tailored instruction and cooperation.

Experiential learning is the process of making meaning from direct experience. Aristotle once said, "For the things we have to learn before we can do them, we learn by doing them." [16] An example of experiential learning could be going to a museum and learning through observation and interaction with an exhibit, instead of reading from a book about others' experiences. Such a method engages learners at a more personal level and, therefore, can be highly effective. It requires no teacher, but, according to David Kolb, an American educational theorist, in order to gain knowledge from an experience, certain abilities are required [16]. Learners must be willing to be actively involved in the experience and be able to reflect on it. They must possess and use analytical skills to conceptualize the experience, as well as decision making and problem solving skills in order to use their new ideas gained from the experience. This way learners not only gain a better understanding of the new knowledge, but they also retain the information for a longer time. Confucius, the ancient Chinese philosopher, said: "tell me and I

will forget, show me and I may remember, involve me and I will understand"[16]. This mode of instruction is typical for video-based games, including *Meet Rembrandt*, where players must navigate game scenarios and make decisions with consequences.

Inquiry-based learning is a form of active learning, where progress is assessed by how well students develop experimental and analytical skills rather than how much knowledge they possess. It is an open learning, with no prescribed target or result which students have to achieve. It means that learners actually think about the results they collect and what they mean, because they are curious what happens, when they perform some action. Inquiry-based learning is a natural mode for many video games, including *Meet Rembrandt*, as players explore, discover and experiment in order to achieve an overall goal.

Self-efficacy can be defined as the belief that one is capable of performing in a certain manner to attain certain goals [17]. In 1997, Albert Bandura, a Stanford University psychology professor, published his book, *Self-Efficacy: The Exercise of Control*. He said that a sense of self-efficacy can play a major role in how people approach goals, tasks and challenges. Those with high self-efficacy expectancies generally perform better in learning and life situations. They are more likely to view difficult tasks as challenges to be mastered rather than threats to be avoided. Believing that they can accomplish a particular goal, there is a higher possibility, that they will persist until the goal is achieved [4]. The stronger the self-efficacy or mastery expectations, the more active the efforts. Low self-efficacy can lead people to believe tasks are harder than they actually are. Bandura thinks, that self-efficacy can be achieved through positive past experiences, reinforcement from the environment, and encouragements from mentors. The main concept is that an individual's actions and reaction are influenced by the actions observed in others. When people see someone succeeding at something, their self-efficacy will increase. Students can watch others playing a computer game and they will think: "If they can do it, I can do it as well". In *Meet Rembrandt*, players are encouraged to keep going, when they get points, go to the next level or succeed in a minigame.

Goal setting involves establishing specific, measurable, achievable, realistic and time-targeted objectives [18]. It is an effective tool for making progress, that allows people to specify their work towards a well-defined goal. All games have goals, what helps students to learn more. In *Meet Rembrandt*, the goal is clearly specified in the beginning of the game.

Such a strategy requires feedback. Without it, goal setting is not likely to be effective. Providing feedback on short-term objectives, helps to sustain motivation and commitment to a goal. Technology can make it easier for teachers to give students feedback about their work. The Jasper Woodbury adventures, mentioned in 2.2, can serve as an example. Initially, teachers had problems finding time to give students feedback, until a simple computer interface was introduced [2]. Using an interactive software program, students could suggest a solution and see simulations of the effects, which had a positive impact on the quality of the solutions that they generated subsequently. Using a computer-based feedback in games can be very effective and often not that hard to implement. In *Meet Rembrandt* players are informed when they do not succeed in a minigame and advised to learn more.

Studies of classroom techniques [3] show cooperative learning results in about a 50 percent improvement over either solo or competitive learning. Some types of games, such as online massive multiplayer, are structured as a team effort toward a common goal. This, however, does not apply to *Meet Rembrandt*, which is designed for a single user.

2.4 Serious Games as a Potential Cure for Poor Education

Games that successfully integrate fun and learning may have potential to motivate young people for learning. They can spark engagement, encourage repetition and practice, motivate learners with challenges and rapid feedback and reach students at various times and places, increasing the time spent on education. Usually, players have a choice and control over whether and how they learn. These environments are also free of the performance demands that students encounter in school. They are often characterized by people's excitement, interest and motivation. Many experiences are designed to capture and sustain players' interest, what can be achieved by interactivity factor.

There are many research-based frameworks for understanding the role of interest and motivation in the learning process. One of them was developed by museum evaluator Deborah L. Perry [1], and even though it was intended for museum exhibits, can be used in other environments, like serious games, as well. The model consists of six components:

1. Curiosity - The visitor is surprised and intrigued.
2. Confidence - The visitor has a sense of competence.
3. Challenge - The visitor perceives that there is something to work toward.
4. Control - The visitor has a sense of self-determination and control.
5. Play - The visitor experiences sensory enjoyment and playfulness.
6. Communication - The visitor engages in meaningful social interaction.

Merrilea J. Mayo mentions five reasons for which video games have the potential to cure poor education [3], including massive reach, effective learning paradigms (discussed in 2.3), enhanced brain chemistry, time on task and learning outcomes data.

Commercial computer games, designed for entertainment, have grown increasingly popular over the past two decades. They can be accessed from video consoles, cell phones and personal computers, both online and offline. Sales of computer and video game software in United States reached \$11.7 billion in 2008 [5] and a recent national survey of young Americans aged 8 to 18 found that their use of video games grew per 24 percent over the past five years, reaching a daily average of 1 hour, 13 minutes. There are, however, disparities in the engagement of different populations. Boys spent usually more time than girls in computer games. To date, only a few investigators have examined this important issue, and the outcomes are interesting. The Entertainment Software Association estimates that in 2004, about 38% of computer game players were women [3]. Certain games, as *The Sims*, can attract even 60-80% of girls. A gaming environment, called *Peeps*, was designed to engage girls in learning computer programming by giving them a possibility to design parts of the game [5]. In the game, players interact with others by dancing with them, and such dance moves can be created by using increasingly complex computer programming skills. The study included approximately equal numbers of ethnically diverse boys and girls and the results suggested that playing the game increased feelings of general self-efficacy among female students. Some recent research suggests that gender differences in interest in games may be diminishing [5].

Learning is most effective when we build on what the learner already knows and using situations with which they are familiar. Young people nowadays are highly visually oriented and much more willing to learn by trying out and watching than reading books. They are able to use computers already at early age. Students play games voluntarily, what increases the likelihood, that serious games could reach them at home and make them spend more time on education.

Recent research shows that the emotions associated with interest are a major factor in thinking and learning. They help people learn, determine what is retained and how long it is remembered. A 1998 study [3] of brain chemistry during video game play found that playing video games stimulates substantial dopamine release. Players' brains showed a steady increase in dopamine levels during play, reaching about twice the amount of non-players. Dopamine is a chemical precursor to the memory storage event, which means that video games may stimulate chemical changes in the brain that promote learning.

In video games we find both visuals and sounds that contribute to entertainment and may contribute to learning as well: directly (by allocating attention to educational content) or indirectly (when increased enjoyment motivates for learning).

There have been several studies done investigating the effects of interactivity on learning among students and the results showed that video games could be a solution for poor education.

One of the studies was done in 2002, when two first year classes diverged in their assignments sets [6]. One of them was a game-based and the other a standard one. After measuring the results and comparing them (using special metrics), it occurred that game-based assignments were showing a significant advantage over the others. They were more complex and required more effort, but still, the students were quite willing to follow them. Also in the successive courses students from the game-based group got better grades (18% improvement).

As an alternative to a game-playing course, there are also game-design courses. One of such was given by Brianno Coller [3]. The students were asked to program virtual cars and a race track, using their knowledge. The other, non-game-based group, consulted textbooks and made exercises. At the end of the year they drew a concept map of what they had learned. It showed that the game design exercises didn't change the breadth of the content learned during the course, but they significantly increased the depth and complexity of what was learned. Also the students were more pleased overall with this course than with others, spending time on their work voluntarily.

At the University of Twente they are developing the new BsC curriculum Creative Technology and decided to adopt a more constructive approach to teaching. A new course was introduced in which students learn "the heart of mathematics" by means of an adventure game called Mr. E's knapsack [7]. It's a game that combines the attractiveness of a fantasy world with education. Players have to actively use mathematical items that they've

picked up in order to solve problems at later stages of the game. Such an approach to learning math is for sure new, but very promising.

In general, technology-based tools can enhance student performance when they are integrated into the curriculum and used in accordance with knowledge about learning.

2.5 Designing a Serious Game

Designing an educational game is a challenge. It is difficult to implement a video game that accurately represents the important content and that is sufficiently engaging to hold students' attention. Narrative is an extremely important feature of games, as it allows students to get engaged and interact with the game without fear of real-life consequences [5]. It makes players feel immersed in the game. However, the game needs to keep a proper balance between the entertainment and the educational content.

In [11], the authors present an educational game *Immune Attack* to show how to build a good serious game. They mention that players usually skip introductory text, so it is wise to allow students to gather information while playing the game. Such content should be presented through clear visual and auditory media that provides information while allowing player's eyes to stay fixed on the action. The desired functionality is to have small assignments, that help students learn. Before each action game a warning is posted to alert players that previously available information is required to win the game. Players who fail to read this information have difficulty winning action games. When losing a game, they are given hints with regard to what information must be learned and encouraged to learn it and try the assignment again until they master the learning objective.

A similar approach is present in *Meet Rembrandt*, and is discussed in more details in chapter 5.

3. Technical background

XIMPEL is an interactive video platform created using the Adobe Flex framework. The aim of this research is to extend XIMPEL with minigames. An important requirement is that authors can include minigames into the XIMPEL playlist in a similar simple fashion as they now include videos.

This chapter describes the XIMPEL platform and illustrates how an interactive video is configured using a playlist file. As the extension to XIMPEL have to be implemented in FLEX, I will also briefly introduce this framework.

3.1 XIMPEL

XIMPEL stands for eXtensible Interactive Media Player for Entertainment and Learning. It is developed at the Multimedia Group within the Faculty of Sciences at VU University in Amsterdam. XIMPEL allows users to create interactive media applications, using the open source Adobe Flex framework, and the open XML description format. It can be used on all computers which have the Adobe Flash plug-in installed, both locally and on the Internet.

XIMPEL provides an open multimedia platform which can be used for both entertainment and education. It enables users to create their own storylines, offering the following features [14]:

- Customizable, clickable overlays and visuals, which can be used to;
- Access different branches of the storyline, and can;
- Link to both external and internal information sources;
- Customizable questions;
- A scoring mechanism to weight the choices made and the answers given;
- Modularity, as XIMPEL can incorporate the user's own (Flash) minigames, questionnaires and other custom media types.

All of the variables, such as the clips and pictures to be shown, the branches, overlays, questions and score points, are modifiable through a collection of XML configuration files, which are read by the XIMPEL Player. Such an approach makes it easy for users to create their own interactive video applications. No programming knowledge is required, because XML files can be easily understood and modified even by inexperienced users.

XIMPEL applications use two XML files. The configuration file allows for simple changes in the settings of the applications, for example, the title screen, score thresholds and messages, or instructions screen. The second file used by XIMPEL Player is the playlist. A basic playlist consists of a number of subjects, containing video fragments, pictures or other media types (including minigames) related to that subject. Video fragments may include branchquestions, that give user the option to switch to different subjects. XIMPEL applications can also count score for rewarding the choice of particular subjects and use description tag to give meaningful information about the possible choices. XIMPEL Interactive Video Format supports refinements, like the indication of time duration and starting time for the overlays, as well as their size and position. Extensive options for adjusting the visual appearance of the overlays include changing text displayed, image, color and alpha attribute. It is possible to create a mouse-over effect by specifying different values for hover attribute.

Below, a declaration is given for *Hendrickje* subject taken from *Meet Rembrandt*:

```
<subject id="Hendrickje">
<description>Hendrickje</description>
<score value="0"/>
<media>
  <video file="hendrickje" repeat="true">
    <branchquestion>
      Which painting do you want to see? (Paintings are clickable!)
    </branchquestion>
    <canvas>
      <overlay width="151" height="243" x="44" y="104" alpha="0" hover_alpha="0.5"
leadsto="WomanBathing" description=" " />
      <overlay width="151" height="155" x="44" y="385" alpha="0" hover_alpha="0.5"
leadsto="Bathsheba" description=" " />
      <overlay width="77" height="35" x="600" y="555" alpha="1" hover_alpha="1"
leadsto="Menu3" description=" " image="m1.jpg" hover_image="m2.jpg"/>
    </canvas>
  </video>
</media>
</subject>
```

Fig. 1 shows the above example, as it can be seen in the browser (two paintings and menu button work as clickable overlays).

To display an interactive screen in XIMPEL Player, we need to create a subject with optional description and score. In the above example, the score is

equal 0, what means that the player will not be awarded with points. The description is shown in the left bottom corner of the screen (Fig.1). Each subject contains various items within *media* tag. In our example, we are using a video file *hendrickje.flv* (the file extension should be omitted in the code, unless it is different than .flv), which is stored in videos directory. It is an image with a text and two paintings, transformed into a video in order to be able to use a branchquestion. The *repeat* attribute set to *true*, means that the user has to make a choice for one of the overlays. Otherwise, the video clip will be repeated in a loop. The branchquestion is shown in the top of the screen and the possible options are hidden in the paintings and a menu button (Fig.1). Various attributes allow for customized configuration of the overlays. In our example both paintings are part of the video and use overlays, which position and size are specified in the code. The alpha attribute set to 0 makes the overlays transparent. Once the user moves the mouse over one of the paintings, the overlays become semi-transparent (*hover_alpha="0.5"*).

The menu overlay is created by using an image of a button (*m1.jpg*), which changes once the mouse is over (*m2.jpg*). It is not a part of the video, and to be fully visible, both alpha and *hover_alpha* attributes are set to 1.

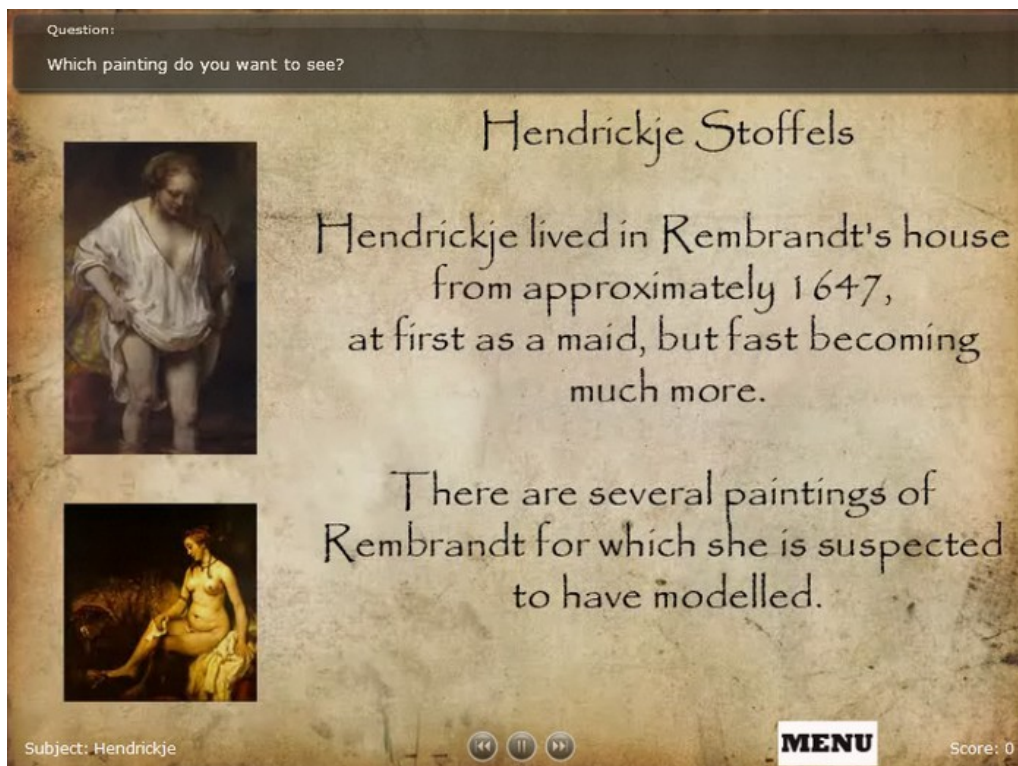


Fig.1

The description tag within overlays can be used to display a text next to the question in the top of the screen to indicate the user's choice. In this example such an option is disabled.

The interactivity in the overlays is achieved by using *leadsto* attribute. The name within quotation marks indicates the subject, which the player is transferred to, after clicking on the overlay. In our example users may choose one of three options: *WomanBathing*, *Bathsheba* or *Menu*.

Besides the built-in media type video (using the tag `<video>`), two other types have been built-in since version 2.0, that include `<youtube>` tag and `<picture>` tag. The work described in this thesis has led to a new media type `<minigame>`.

More detailed description, including tutorials, can be found at ximpel.net

3.2 Adobe Flex, MXML, Action Script 3.0

Adobe Flex is a software development kit (SDK) released by Adobe Systems for building expressive mobile, web, and desktop applications based on the Adobe Flash platform. This free, open source framework allows creating applications that share a common code base, reducing the time and cost of building and maintenance. Flex provides a modern, standards-based language and programming model that supports common design patterns. MXML, a declarative XML-based language, is used to describe user interface layout and behaviors. Interactivity is achieved through the use of ActionScript. The Flex SDK comes with a set of user interface components including buttons, list boxes, data grids, trees, text controls, and various layout containers. Flex can be used to create web applications that run in the browser through Adobe Flash Player software, or desktop applications that can be used even when disconnected from the Internet.

ActionScript is the programming language of the Adobe Flash platform. Originally developed as a way for developers to program interactivity, ActionScript enables efficient programming of Flash Platform applications for everything from simple animations to complex data-rich, interactive interfaces. ActionScript code must be compiled into an SWF file for playback in one of Adobe's Flash client runtimes. Flex projects can be created using Adobe Flash Builder and run in order to create an SWF file. In addition, there is also an option of using the free command-line compiler *mxmlc* [8, 9, 13].

In this project Adobe Flex 3 with ActionScript 3.0 is used.

4. Minigames

The aim of this research is to extend XIMPEL with minigames. This chapter focuses on the technical implementation and the presentation of a number of templates that I have created.

4.1 Introducing Minigames in XIMPEL

Initially, XIMPEL supported only one media type - video, and allowed for adding custom media types, defined by the user. With new versions, more media types have been added, including pictures, Youtube videos and UMap (a media type which allows using geographical maps within XIMPEL). It has been always possible to incorporate user's own games, but it required significant programming knowledge in Flex. Introducing a new media type, which allows for adding minigames to the playlist in a similar fashion as any other media types, makes it easier for designers to include small games in their applications.

To program a custom media type, one needs to create a class, that implements the *IMediaType* interface. In order to use minigames in XIMPEL, *Minigame* class has been created (the code of *Minigame.mxml* can be found in Appendix A) and registered in the main application:

```
var minigame:Minigame = new Minigame();
minigame.addEventListener(MediaScoreEvent.SCORE_RESULT,updateMediaScore);
myXimpelPlayer.registerMediaType(minigame);
```

Every time a new minigame is added to the playlist, a new object is created with its local connection, that enables sending the status and score of each minigame to the main application. Now, using minigames in the playlist is as simple as with any other media type. Below I present an example (taken from *Meet Rembrandt*), how to include minigames in the playlist using `<minigame>` tag:

```
<subject id="HouseTest" leadsto="Studio">
  <description>Rembrandt's House</description>
  <score value="0"/>
  <media>
    <minigame file="housematch.swf"/>
  </media>
</subject>
```

There are two ways to create minigames. The users can either program their own games in Flex, or use a game template. The first option requires programming knowledge and adjustments in the code, so that the games are compatible with XIMPEL Player. More details can be found in the tutorial in Appendix B. Game templates allow for quick and easy creation of minigames, even for inexperienced users. In this research, I have created four game templates. Authors can customize them for their applications and include in their playlists. Each template allows customization by modifying XML files and adjusting image files. More detailed description of how to configure the templates can be found in Appendix B.

4.2 Minigame Templates

I have created four templates: a memory game, a sorting game, a matching game and a questionnaire. All of them are simple, but very flexible and can be customized according to users' needs. In this chapter I present my templates and discuss their role in the playlist and how they aid learning. To better illustrate my words, I show examples from *Meet Rembrandt*.

4.2.1 Memory Game

In a memory game the player needs to find matching images. The score is dependent on total clicks, motivating players to try harder. The template is universal and gives the possibility of using any number of cards. It can be easily customized by changing files in the *assets* directory and modifying their names in an XML configuration file. XIMPEL Player reads the dynamic XML file in order to place all the pictures on the screen in a random order. The

example in Fig.2 shows a memory game from the final test of *Meet Rembrandt*, where different paintings by Rembrandt are to be paired.

Memory game can be used in the playlist in order to help players remembering important items. Even though the task is not complicated, it can support studying by stimulating visual memory. The user sees pictures a few times before the pairs are found, and such repetitions can have a positive effect on learning. Students get more acquainted with the topic, and, in the case of *Meet Rembrandt*, can recognize Rembrandt's paintings. Memory game is an example of a minigame, where not the knowledge is being checked, but it rather serves as an experience, from which players can learn.

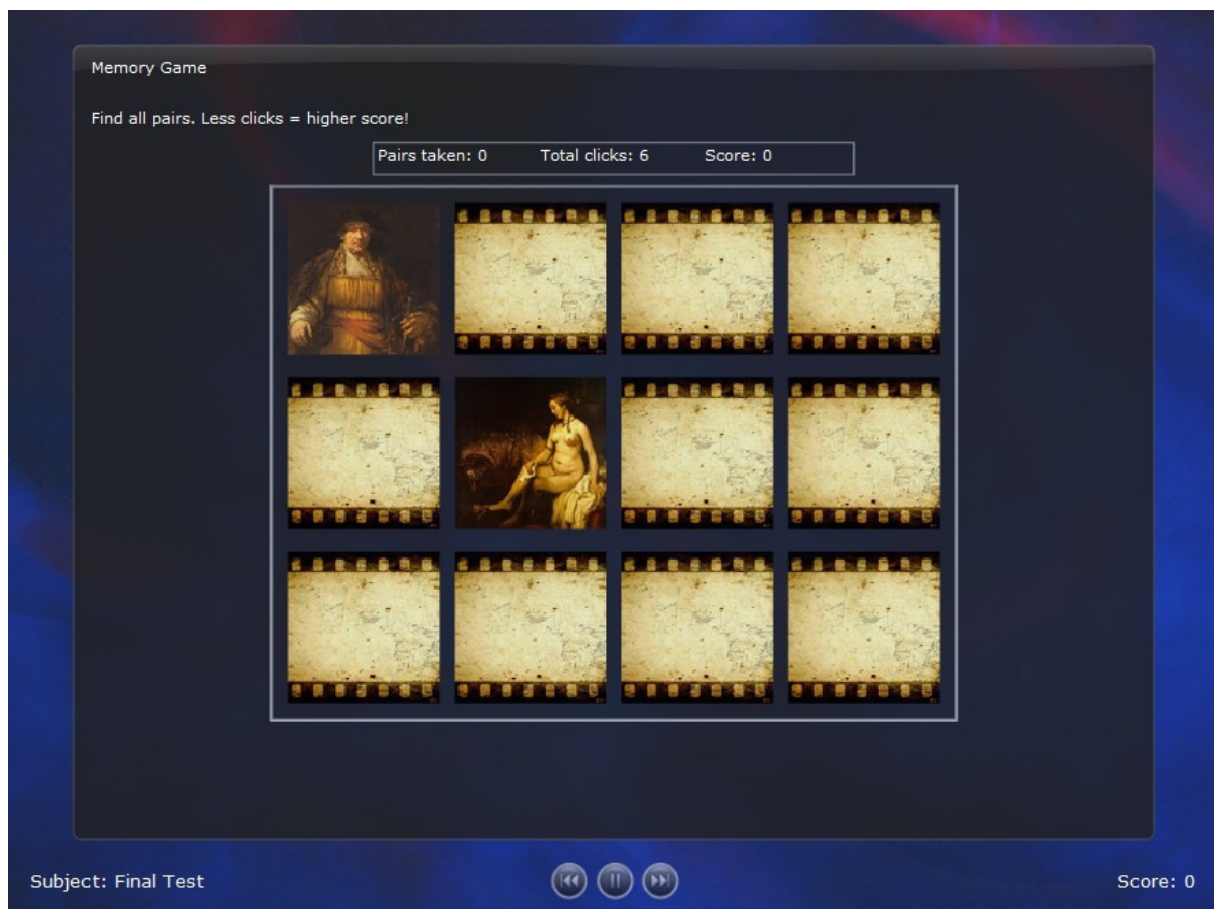


Fig.2

4.2.2 Matching Game

Matching game is a simple assignment, in which players need to find matching pairs. It is, however, different from a memory game, as here all the images are uncovered from the beginning and the pairs are not the same

images, but two different items, that have something in common. They can be easily changed by adjusting XML files. The number of pairs is flexible and they are randomly placed on the screen. Fig.4 shows an example of a minigame, where players need to match Rembrandt's paintings with their names by drag-and-drop method. Here also the score is dependent on total moves, encouraging players to think, instead of only trying out all the possibilities. This, however, can be used in different ways. The matching game can work as a knowledge test, where students try to get the best results, but it can also serve as an experience because of two reasons. It supports the inquiry-based learning paradigm, where students learn by just trying out and checking which images would match. It lowers the score, but helps to memorize the content. The other reason is that often some of the pairs are obvious. For example, in Fig.3, the player does not need any knowledge to find the matching painting for "Saskia in a Red Hat" name. It may give the feeling, that the task is too easy, but it still requires some brain exercise, which helps to remember the content.

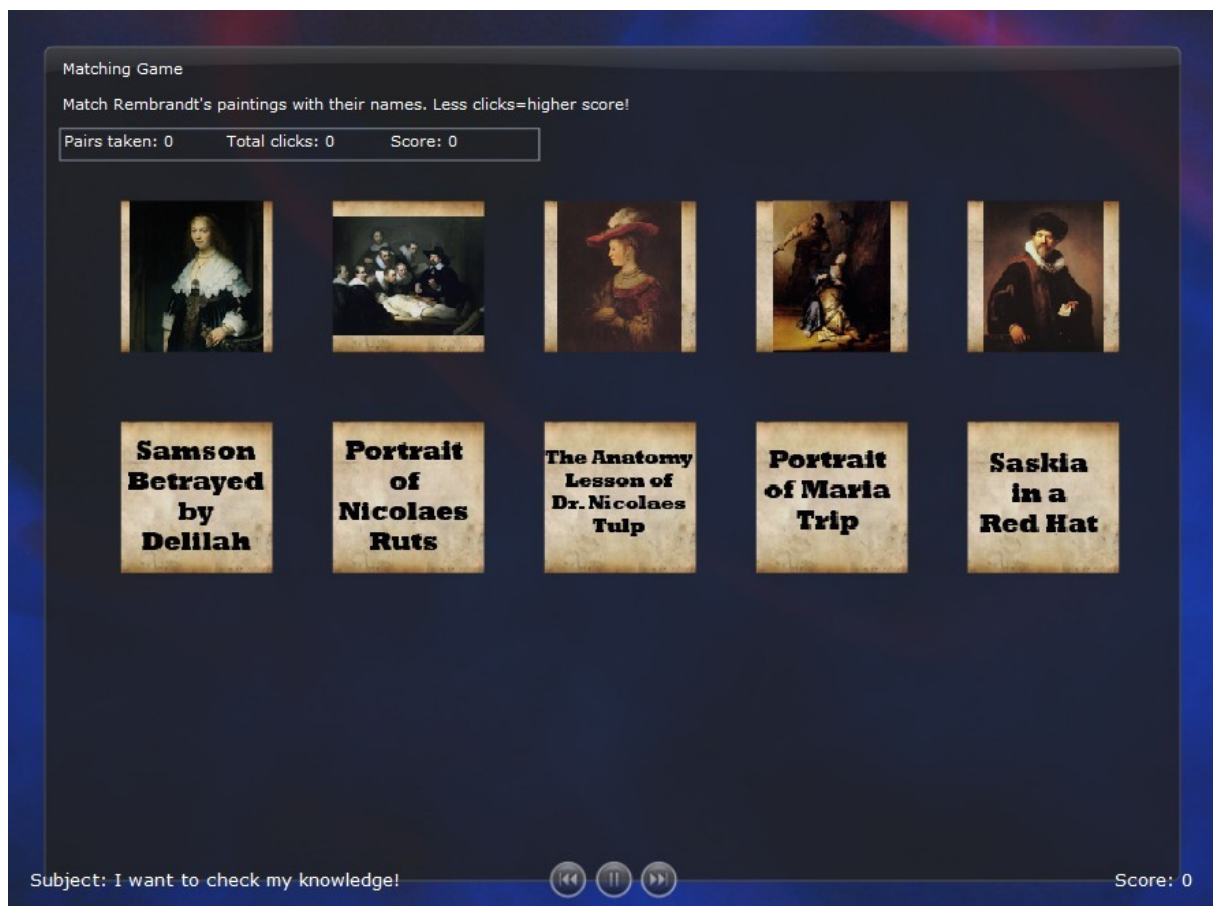


Fig.3

4.2.3 Sorting Game

Sorting game is a minigame, where players need to sort images in a correct order. It works very similar to matching game, as it also uses drag-and-drop method, the score is dependent on total moves, and can serve both as a knowledge test and a learning experience. The template can be easily customized by changing XML files. The number of images is flexible. Fig.4 shows an example from *Meet Rembrandt*, where the task is to sort Rembrandt's paintings from the oldest to the newest one.

Sorting game is a powerful tool for checking student's knowledge, but can be also used for learning, when the paintings are being dragged one after another, and this way players check what the order should be.

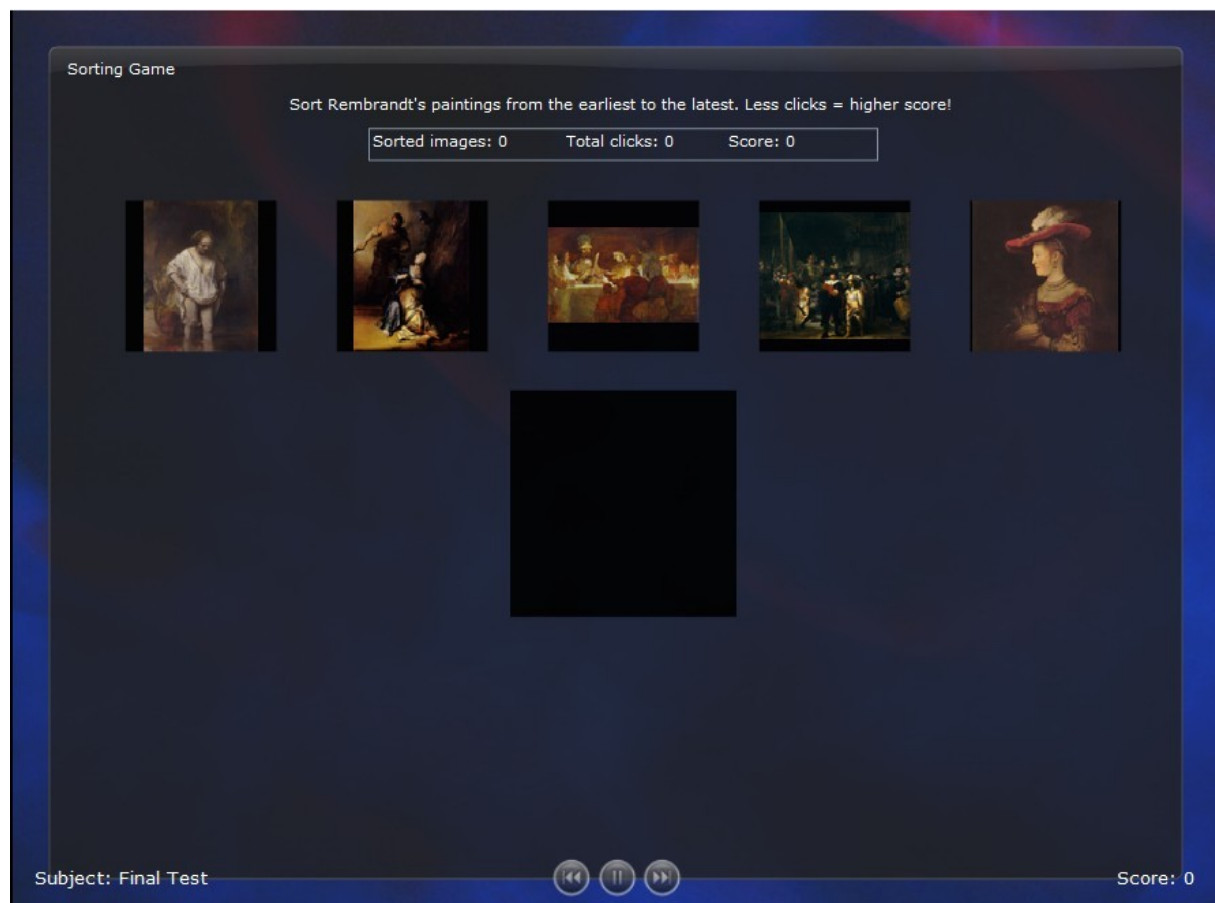


Fig.4

4.2.4 Questionnaire

The last of existing templates allows for creating questionnaires, that can be used in the playlist as a knowledge test. The number of questions and answers is arbitrary, however, only one answer is correct. The content of the test can be easily modified in XML files. In opposite to sorting and matching games, questionnaires do not allow for checking the correct answers by trying out different possibilities. The results are being analyzed only after the user has clicked the *Submit* button. It means that these minigames can be used in the playlist only for examining the knowledge. It is, however, a powerful tool, which flexibility allows for creating any kind of tests. Fig.5 shows an example from a final test about Rembrandt.

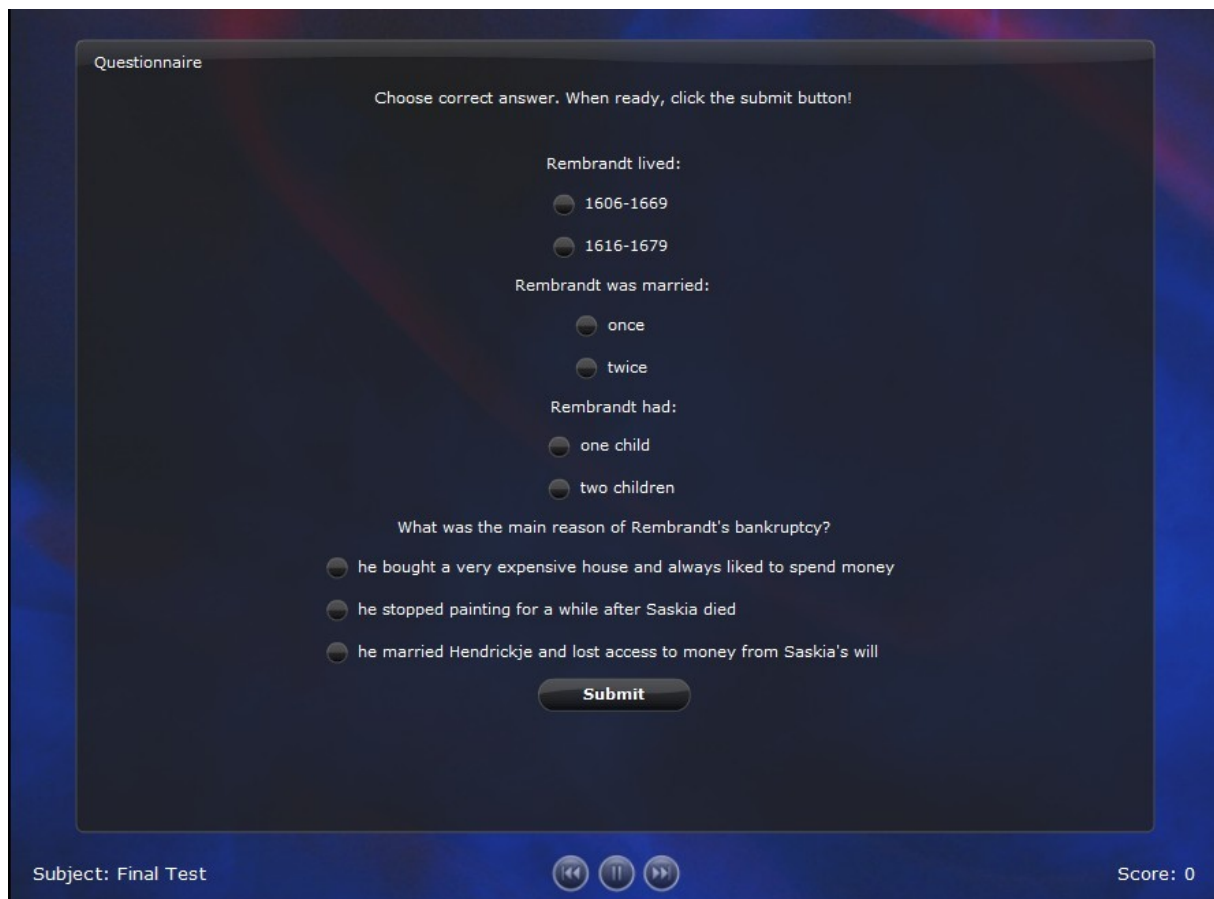


Fig.5

5. Meet Rembrandt with Minigames in XIMPEL

Meet Rembrandt is an educational interactive video game, that I have created in order to show the potential of minigames in XIMPEL. It is designed in a similar way as most of the games, where players gain points and go to the next levels in order to achieve an overall goal. In case of *Meet Rembrandt*, the story starts when a student (player) fails his art exam and is being sent to the seventeenth century by his professor. To come back, he needs to talk to different people, make decisions and learn about Rembrandt in order to pass small assignments, that allow him to go to the next level. After finishing all the levels, the player is given a final exam checking his knowledge about the famous painter. Throughout the game, students learn about Rembrandt's youth, his family, paintings and other details of his life. Everything is presented in the form of short video clips. Most of them have overlays, that force players to make their decisions about what to do next. A very new approach, that has not been previously introduced in any other XIMPEL production, is the occurrence of minigames. To go to the next level, players need to pass them with a sufficient score.

XIMPEL's applications use playlists, that can be depicted in a form of a storygraph. A storygraph is a tree, that represents the design of the game: all the media items, branches, decision points, etc. I use the storygraphs as a guideline to explain the three parts of my game. Various colors and shapes emphasize the differences between the items. The videos are divided into three groups: a regular video (blue rectangle), a video with a branch question, where a decision about the next move has to be made (turquoise rectangle), and an obligatory video (pink rectangle). There are also notifications (white rectangle) to give players important information about the gameplay and "Did you know?" boards (yellow rectangle), which contain interesting facts, that have not been presented in the videos. Minigames are shown as green ovals. Octagons occur in the second and third part, when the menus are introduced.

5.1 Part One: Getting To Know Rembrandt

The game starts with a video showing an exam and the professor telling the student, that they have failed. In the graph shown in Fig 6. this video is represented by the first blue box. In the next clip we can see the seventeenth-

century Amsterdam and the player is asked if they want to check their pocket or talk to a passer-by. In the graph this choice is shown by the turquoise rectangle, representing the video with the two overlays and the arrows pointing to the box with the passer-by video and the pink box with a letter.

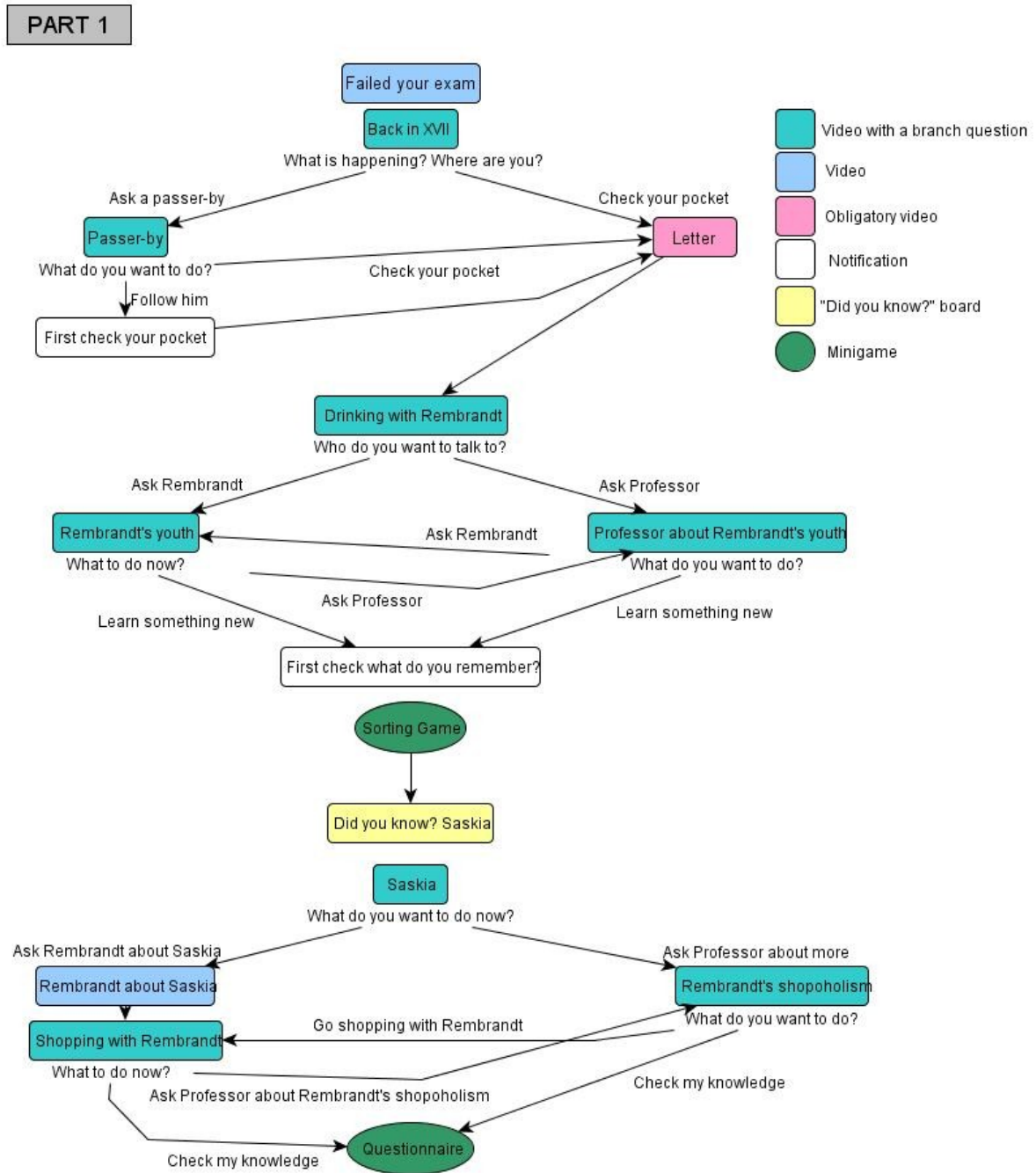


Fig.6

If the player chooses to talk to the passer-by, they can see a video, in which a man explains, that they are going to meet Rembrandt. The player can decide, if they want to follow the passer-by, or check the pocket, but in the first case, they are informed, that the other activity is important. This is depicted in the storygraph in form of a white notification box. It does not matter, which option has been chosen, because all the branches in the graph end up at the same, obligatory video (a pink box). There, the player can read the letter explaining, that they had been sent to the seventeenth century in order to learn about Rembrandt. They are told, that they can talk to people, ask professor for help and reach next levels of the game, until they have gained enough knowledge to be sent back to the future.

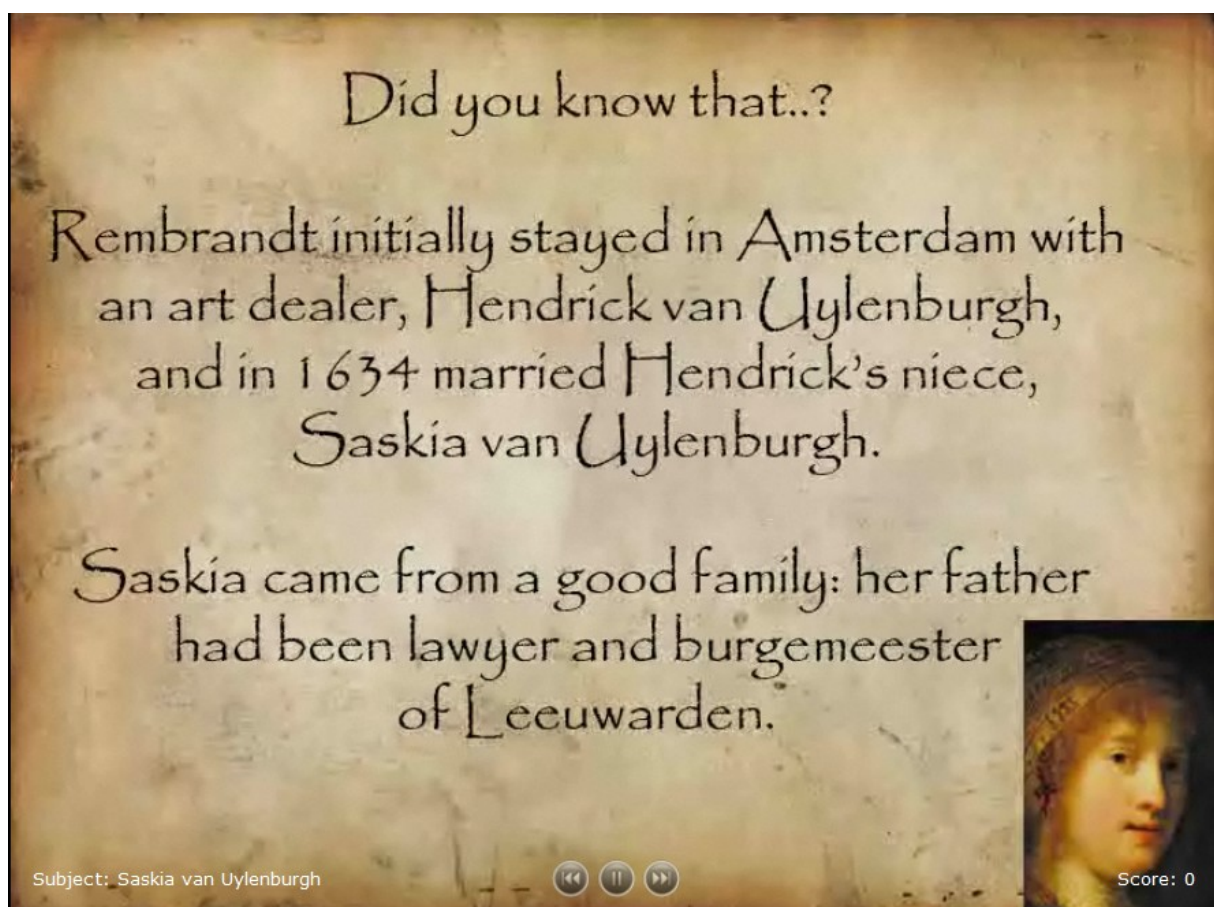


Fig.7

After getting to know what is the goal of the game, players find themselves in a place, where people are drinking with Rembrandt. Here, the player gets an opportunity to talk to Rembrandt about his youth. It is also possible to ask the professor. It does not matter which option has been chosen, as it is possible (but not obligatory) to see the other video as well. In the graph this is represented by the arrows pointing in both directions (from the

turquoise box with the video about Rembrandt's youth to the box with the video with the professor, and the other way). After this section, the first minigame occurs, where players need to sort events from Rembrandt's life. The minigame in Fig.6 is represented by a green circle .

The next section is about Saskia. There is a "Did you know?" board (a yellow box in Fig.7) with information about Rembrandt's wife, which is followed by a video clip about her. It contains a branch question, which gives an opportunity to ask Rembrandt about Saskia, or learn something new from the professor. If the first option is chosen, the player listens to the painter and is invited to go shopping with him. Afterwards, they can either check their knowledge, or ask professor about Rembrandt's shopoholism (which leads to the same video as the other option from the previous branch question). After this section, there is a questionnaire checking the player's knowledge.

5.2 Part Two: Rembrandt's Triumph

The second part starts with a menu, which allows for jumping to different sections (Fig.8). The player can choose, if they want to start from the beginning, the section about Rembrandt's house, in Rembrandt's studio, or the part about the Night Watch (Fig.9). When one of the later sections is chosen, it is not possible to come back to the earlier points.

In the beginning, the player is informed, that they moved to 1639, when Rembrandt bought his new house. In the next video the professor talks about the house and players can choose if they want to go see the kitchen or go for a tour through the house. If they decide on the first option, they are informed, that the knowledge from the tour is necessary in order to pass the next minigame. It is up to them, however, if they will go on the tour. In the next step they need to match pictures of Rembrandt's house with their descriptions.

The next section presents Rembrandt's studio. Here, players can check one of the five paintings or ask the professor how to paint portraits (Fig.10). Each option gives the possibility to come back to Rembrandt's studio and check more paintings. There is also an instant option to check knowledge (both in the studio and when watching the paintings). When this is chosen, the player is presented with four minigames, that need to be passed with a sufficient score in order to go to the next level. If the player does not get enough points, they are informed about their poor result and sent back to Rembrandt's studio.

PART 2

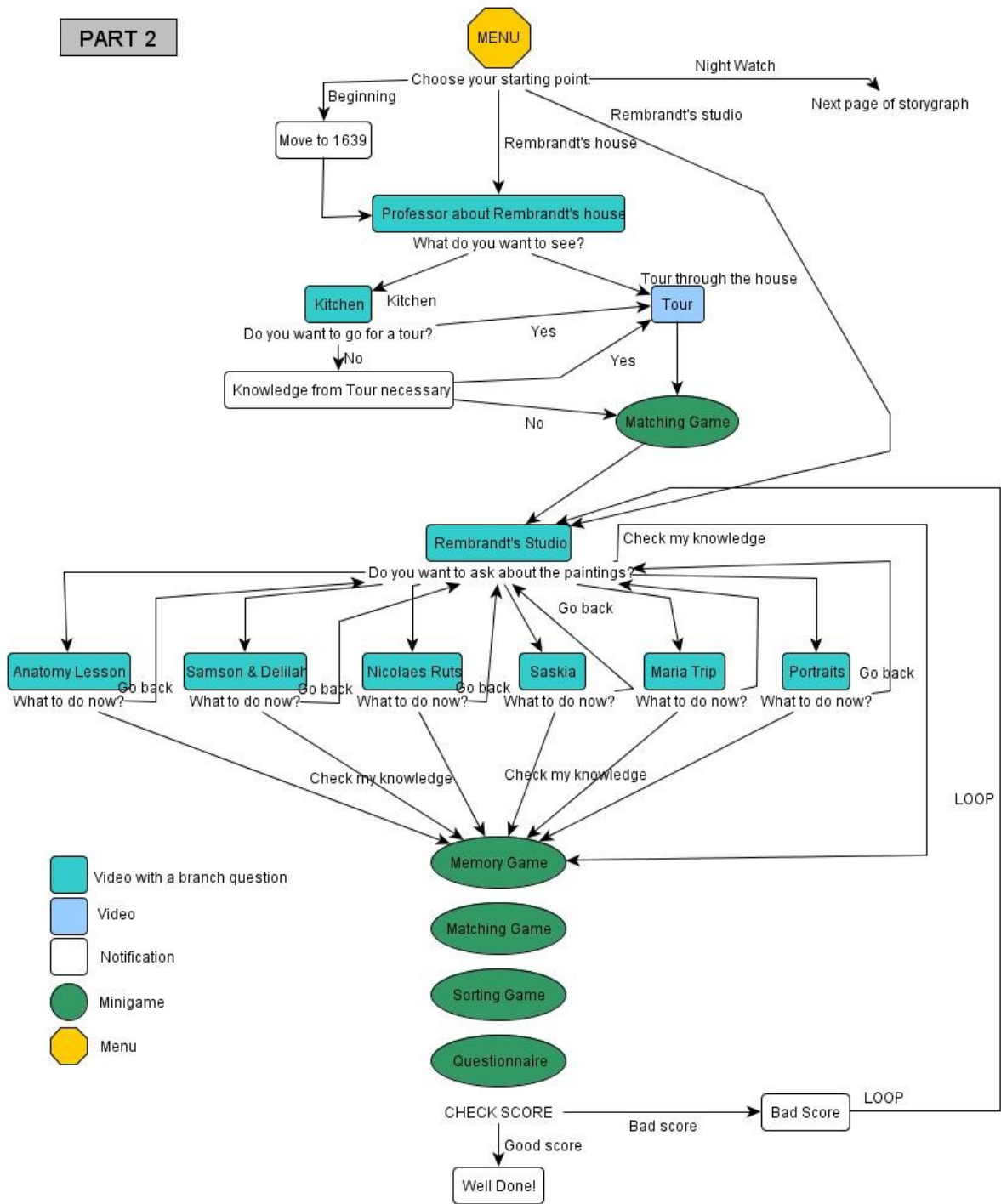


Fig.8 (continuation on the next page)

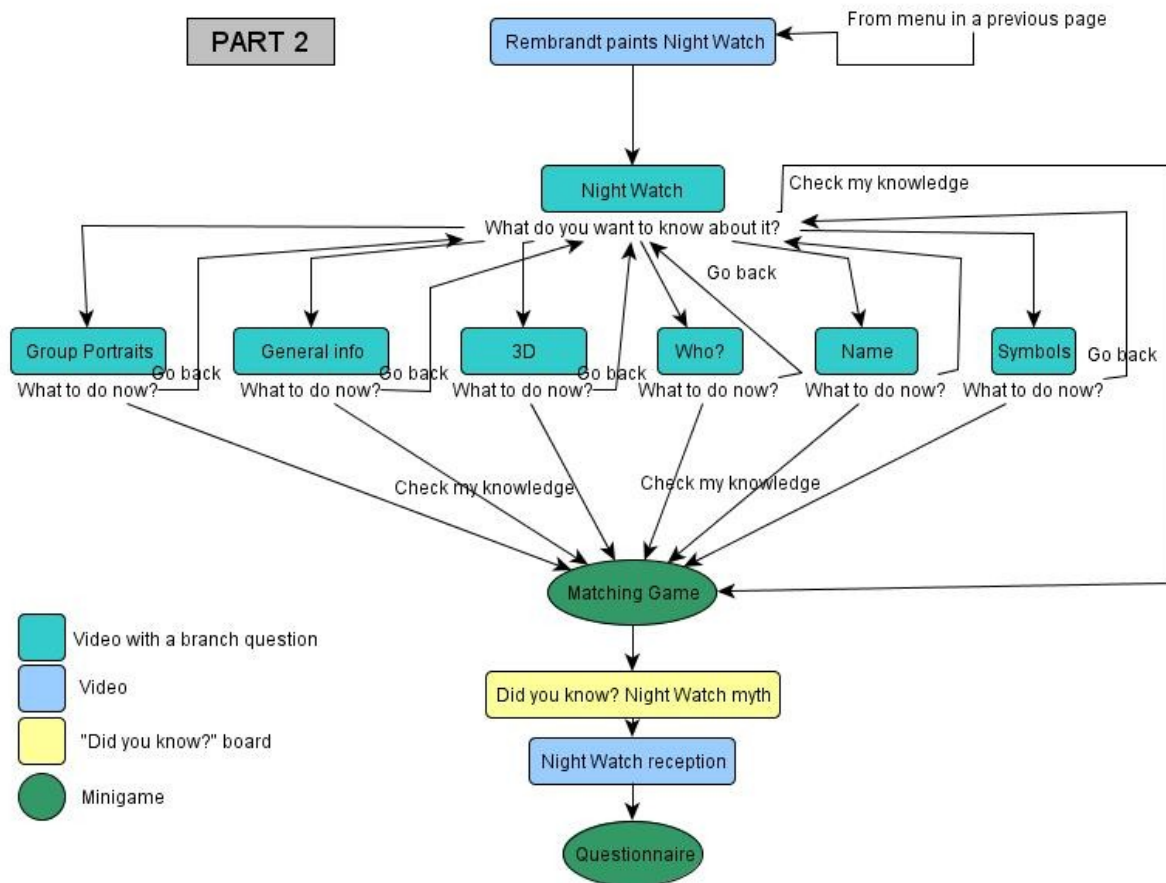


Fig.8 (continuation)

The next section presents Rembrandt's studio. Here, players can check one of the five paintings or ask the professor how to paint portraits (Fig.10). Each option gives the possibility to come back to Rembrandt's studio and check more paintings. There is also an instant option to check knowledge (both in the studio and when watching the paintings). When this is chosen, the player is presented with four minigames, that need to be passed with a sufficient score in order to go to the next level. If the player does not get enough points, they are informed about their poor result and sent back to Rembrandt's studio.

The next level starts with a video of Rembrandt painting the Night Watch, followed by a board with the painting, which offers a multiway menu, similar to the one from Rembrandt's studio. The player can ask about different aspects of the Night Watch and gets the possibility to check their knowledge at any time. The test is in a form of a matching game, where different parts of the painting are to be paired with their descriptions. After the game, players are presented with an additional "Did you know?" board and a video, explaining the myth about the painting. The second part of the game ends with a short questionnaire about the Night Watch.

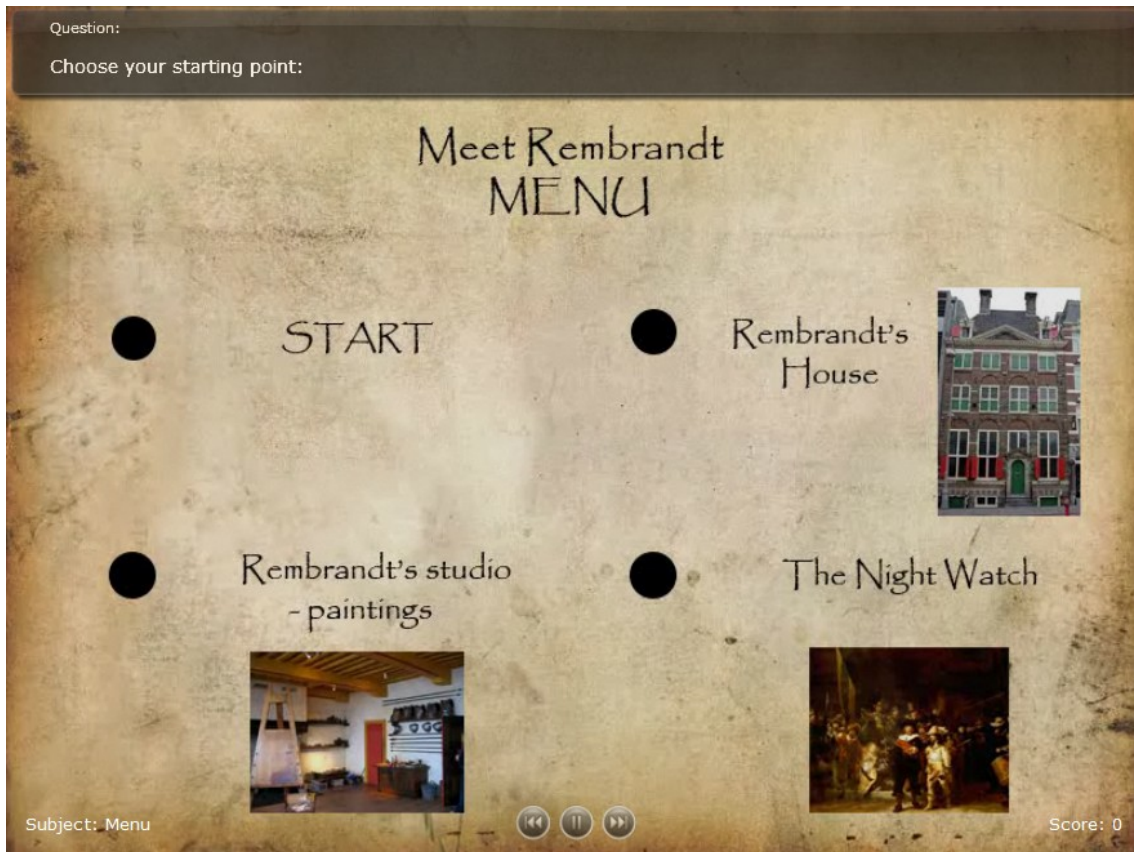


Fig.9

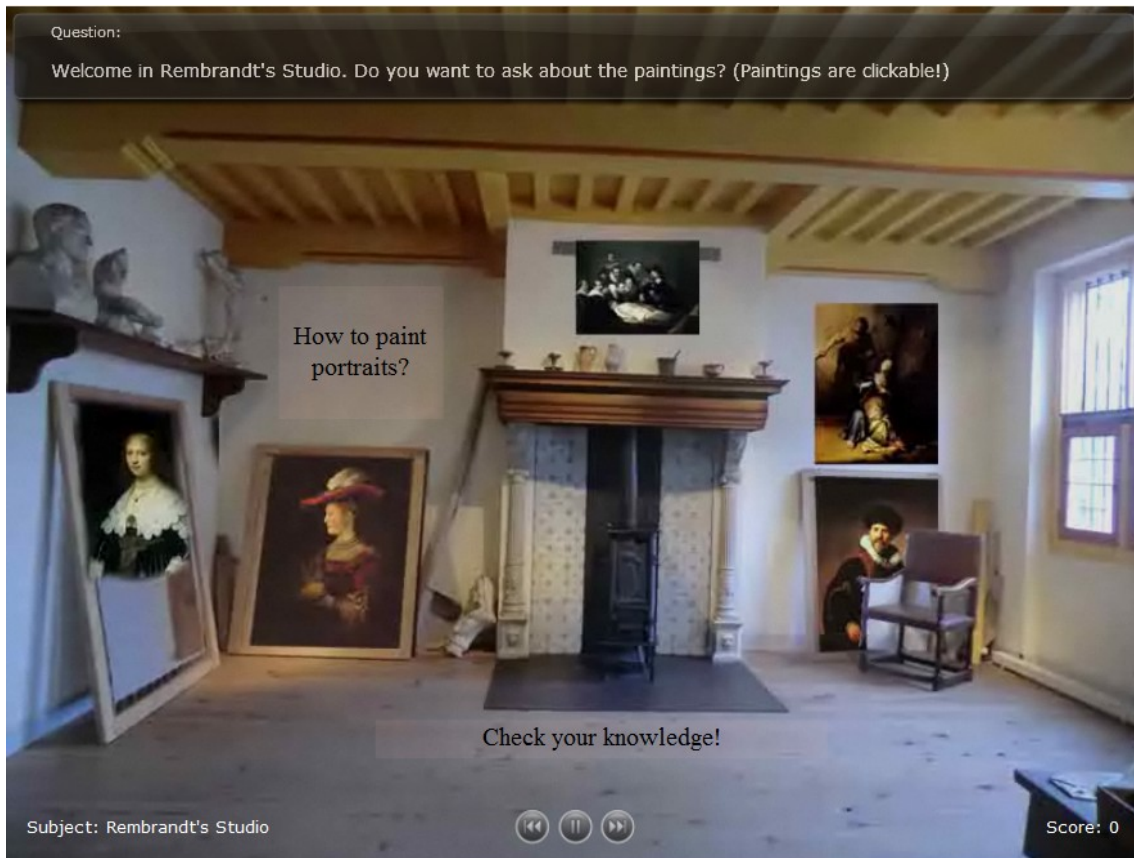


Fig.10

5.3 Part Three: The Twilight

The third part of the game introduces a complex menu (Fig.12), which allows for unlimited jumping within different sections. As a starting point, the player can pick one of the seven subjects or travel in time (by choosing a year from the timeline menu) (Fig.11). The menu is accessible throughout the entire game, which is presented in the storygraph (Fig.13) in a form of a beige octagon. The yellow and purple octagons (with names from M1 to M7 and from T1 to T7) in the storygraph mark the points, where the player is sent after choosing one of the starting positions from the menu (Fig.12).

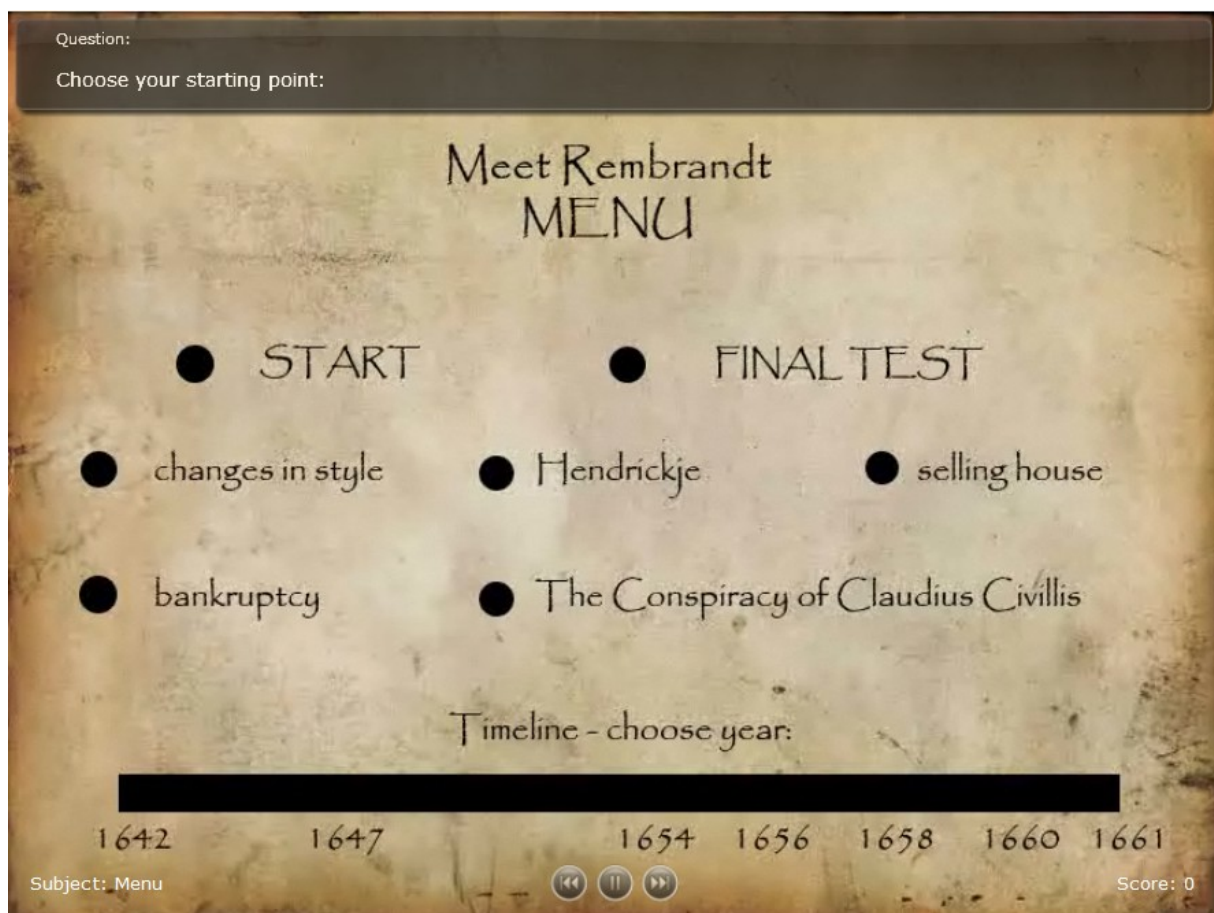


Fig.11

The third part starts with a few video clips presenting Rembrandt's problems, Saskia's death and changes in the painter's style. In the last of the movies, there is a branch question, which allows the player to ask about the changes in the style. There are three different options and each of them gives the possibility to come back and see the remaining videos. There is also an instant option of checking the knowledge by means of a matching game.

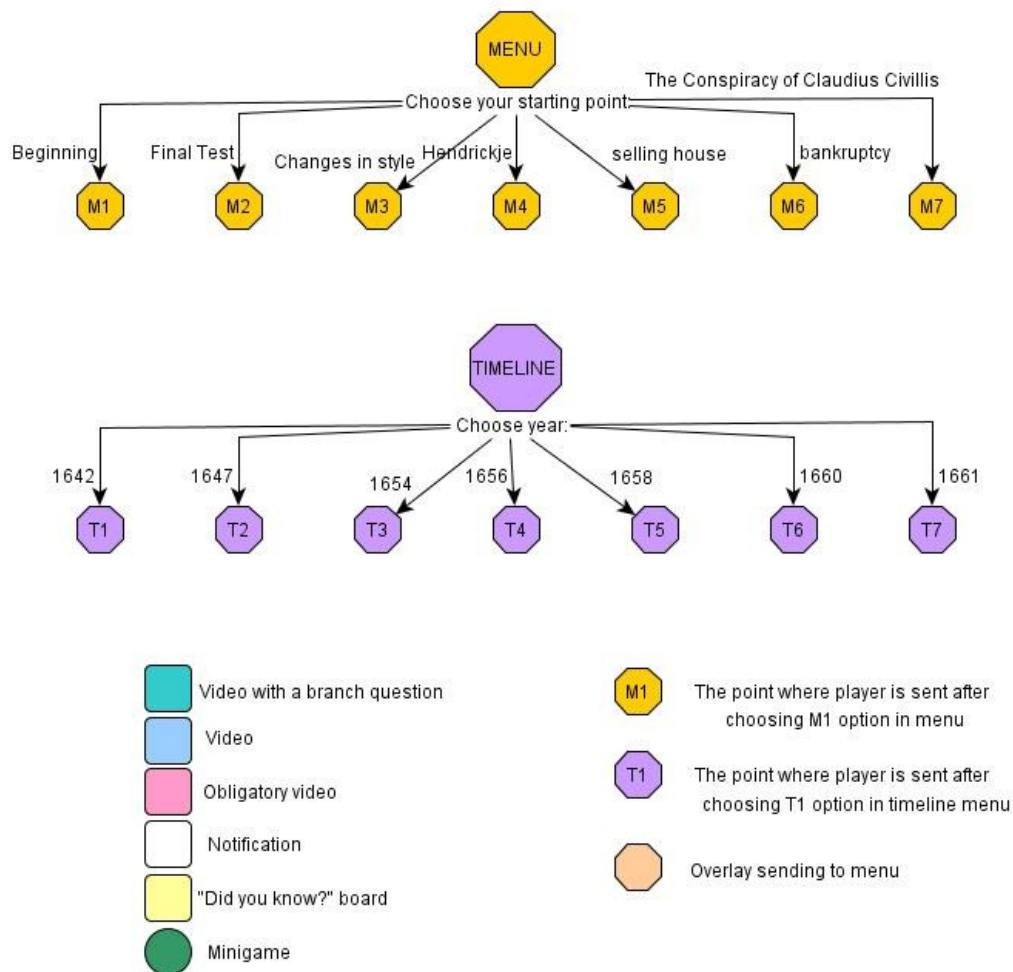


Fig.12

After the minigame, the player is informed about moving to 1647, where they can see a video with Rembrandt painting his maid. They can watch two paintings, for which Hendrickje has modeled. Afterwards, there is a "Did you know?" board about their daughter and a video explaining why Rembrandt and Hendrickje could not get married. It contains a branch question, where players need to decide if they want to see what people say, or check what Church Council is thinking. Both videos can be watched, giving also the possibility to check the knowledge in a simple questionnaire.

The next section starts with a notification, that the player has moved to 1656. Here, they can see the video, where Rembrandt's house is being sold and then go have a look at the house or ask the professor about the situation. Afterwards, they can watch Rembrandt's self portrait and play a minigame, where events from Rembrandt's life are to be sorted.

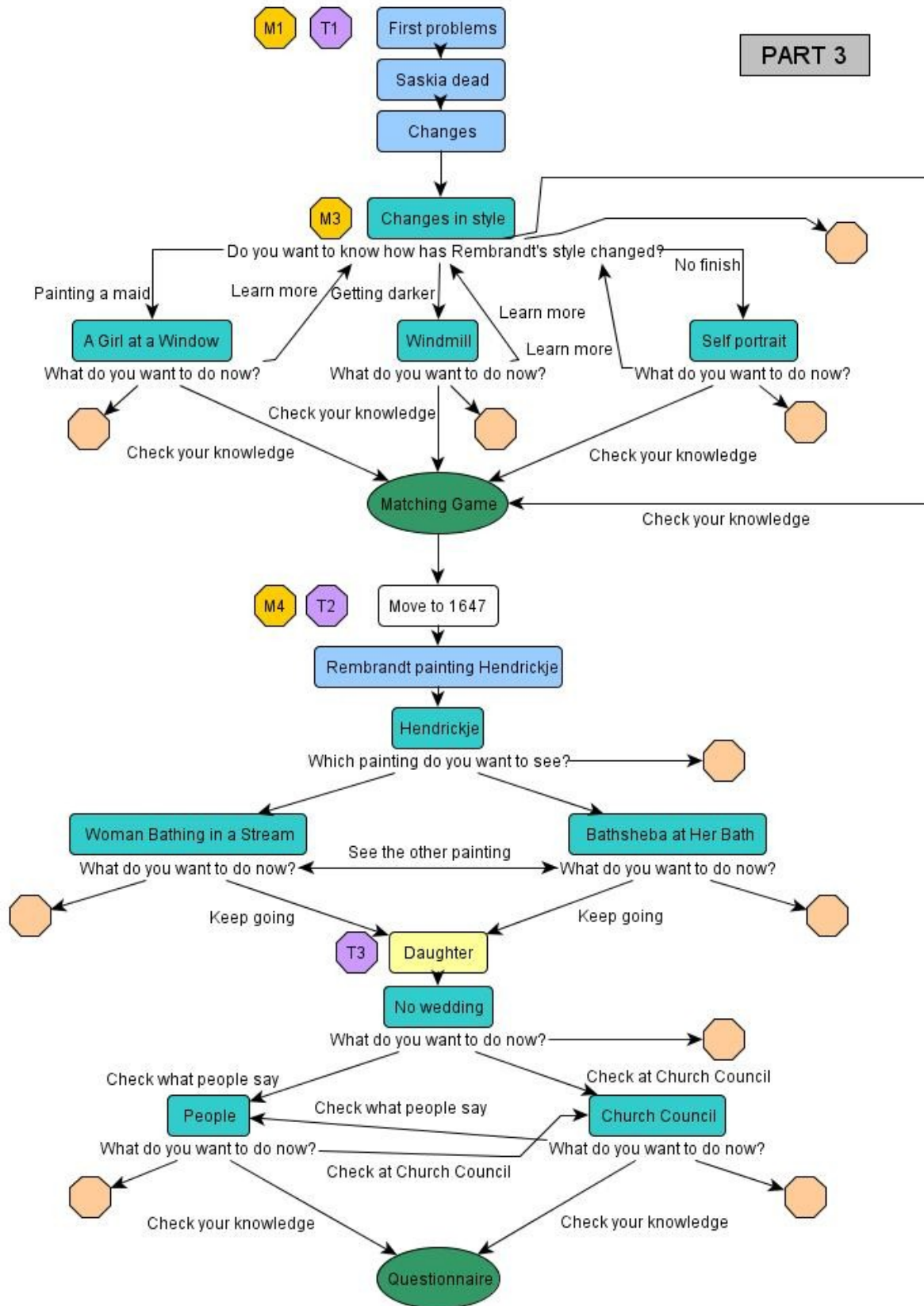


Fig.13 (continuation on the next page)

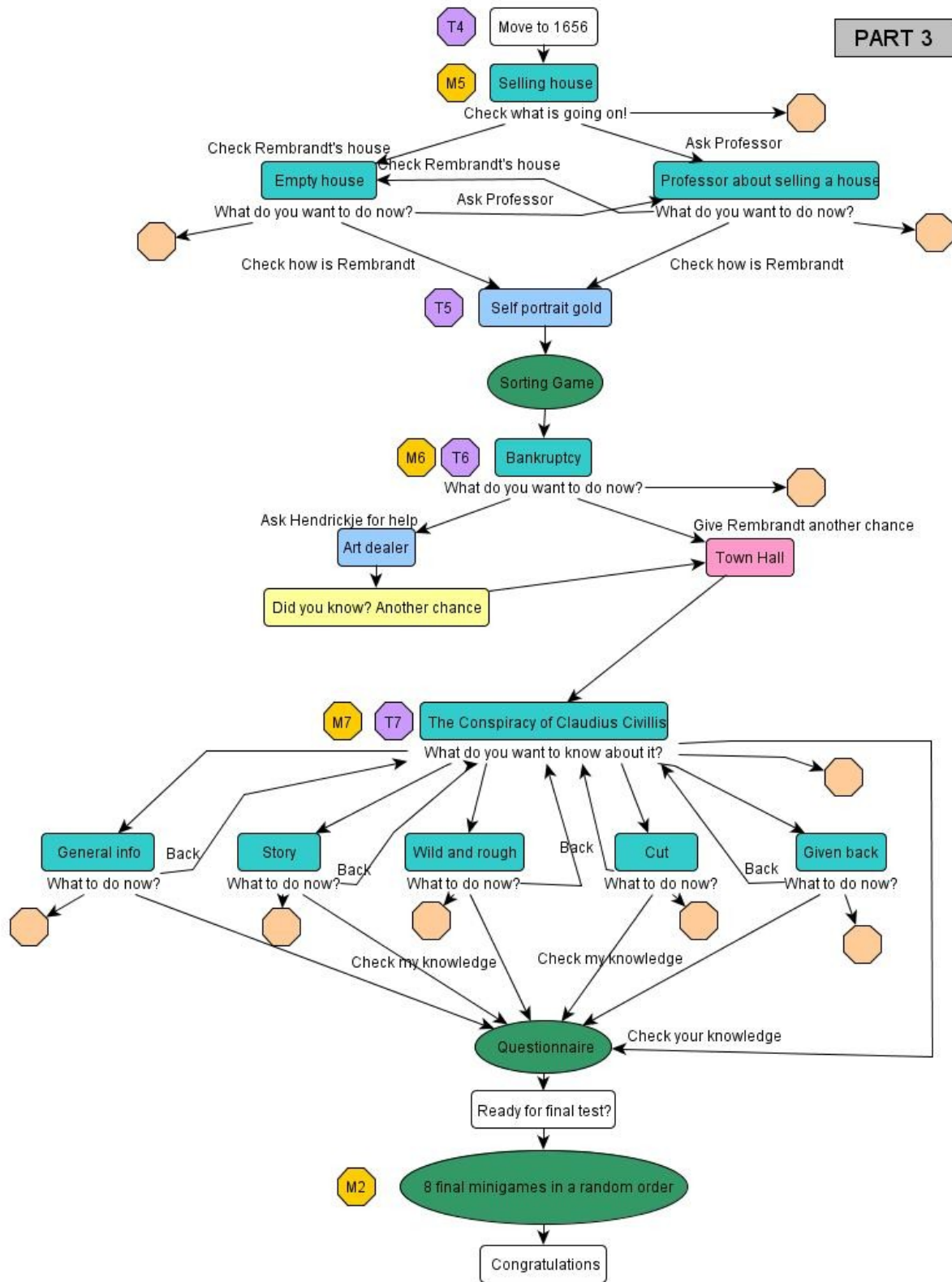


Fig.13 (continuation)

In the next video, the player is being informed about Rembrandt's bankruptcy and is given the choice if they want to ask Hendrickje for help, or give Rembrandt another chance. When the first option is chosen, players see a video explaining how Hendrickje decided to start an art dealer business. Next, there is a "Did you know?" board, which informs, that Rembrandt had been given another chance. Both this, and the other branch from the previous question, end up at a clip about the Town Hall, where a new painting by Rembrandt was to be hung. The last section is about The Conspiracy of Claudius Civillis, which presents the painting with a multiway menu, as in the second part of the game. Here, the player can ask about five different aspects of The Conspiracy of Claudius Civillis, and is given the possibility to check their knowledge by means of a questionnaire about the painting.

In the end, the player is informed, that they are about to take the final test. The test consists of eight different minigames (a memory game, three sorting games, two matching games and two questionnaires) played in a random order. The game ends with a congratulations screen.

6. Designing a Serious Game. Three versions of *Meet Rembrandt*.

In many aspects *Meet Rembrandt* follows the structure of an educational game *Immune Attack*, presented in chapter 2.5. There is a story behind the game, which is introduced in the first video, and the goal, which is explained at the beginning of the game in form of a letter to the student. Players gather information throughout the game by watching videos, clicking on different overlays and making decisions. The idea is to create the feeling that Rembrandt is still among us, alive, and that we can talk to him and hear the story of his life. The game uses visual and auditory media to present the educational content. Minigames play the role of action games, which serve as an experience, from which students learn, or check their knowledge. Players must succeed in minigames in order to go to the next level.

There are several aspects that can be designed in different ways and impact on the learning effect. To explore these options, I have created three parts of my game, in which I have made different design decisions. In this chapter I discuss the possible approaches and its consequences, comparing the three parts within *Meet Rembrandt*. In the next chapter I will discuss the results of a user study, that I have performed in order to test the effects of my design decisions.

6.1 Branching Structure

XIMPEL allows for having any kind of branching structure, including binary trees, multiway menus and loops. The first part of *Meet Rembrandt* (Fig.6) mostly uses a binary option, where the user can choose to watch one of the two videos. They are also given the possibility to watch the skipped video. Both clips are presenting the same information, but in a different way - one is told by Rembrandt, the other by the professor. The player is not forced to see both videos and does not miss any important information when skipping one of them. Watching both, however, might help in memorizing the content. Constant binary choice throughout the game might give an impression of a boring routine, but it is consistent in its structure.

The second part (Fig.8) uses a multiway structure with around seven options to choose. After watching each of the videos, the player can decide if they want to see the other clips as well, or take a knowledge test. Having all

the videos within one main screen might be a more clear indication of what is necessary in order to pass the assignment. On the other hand, it might make players want to "keep going" and make them tired of so many videos to see.

The third part (Fig.13) is not consistent in its structure, what might be more interesting for the player, but also more confusing. It uses four different approaches: a multiway structure as in part two (with six options), a multiway structure with three options (which is similar to the previous one, but probably less tiring for the player) and two different binary structures. One of them is similar as in part one of the game, but the information in two videos is different, which means that both need to be watched in order to pass an assignment. The player is given the possibility to come back and watch the other clip, but is not forced to do so. The last approach in the third part is a binary structure with no possibility to watch the skipped video. This construction, however, is used only when the other video is not necessary to win the game.

6.2 Minigames

The most important aspect when designing a serious game is to have a good balance between the entertainment and educational purposes. Videos in *Meet Rembrandt* convey a lot of useful educational content, but it might be hard for students to memorize all the information by just watching them. Storytelling can be entertaining, but learning is much more effective, if it involves some active engagement that is enjoyable for students. In *Meet Rembrandt*, minigames play such a role. It is very important to find a good balance between played videos and minigames, so that the players have fun as in any other commercial game, but also acquire knowledge. The three versions of *Meet Rembrandt* use different approaches.

In the first version, the storytelling is predominant. There are only two minigames in opposite to a lot of videos. As an effect, players might feel that there is too much storytelling and they lose their focus. Knowledge is not checked often enough to make students immerse in the game and memorize all the information.

In the second and the third version of the game, minigames appear after each segment has been finished. This way students do not get an opportunity to forget some information. There is an (almost) permanent option of testing player's knowledge. It means that the users have more control over it, but

might lead to a situation, when they attempt to play a minigame without gaining enough knowledge.

At the end of the game (part three), there is a final test, which covers information from all three parts of the demo. It needs to be passed in order to achieve the goal of the game. It consists of eight different minigames, that check player's knowledge from different perspectives. There are two ways to implement it in XIMPEL. One of them is to create one screen with eight overlays, that allows players to choose the order of playing. After each minigame, the player would be sent back to the starting screen and, if the last game had been passed, that overlay would be disabled. This option is not implemented in *Meet Rembrandt* because of technical reasons (XIMPEL constraints). The other way is to have all eight games one after another. Here player has no impact on the order of minigames. Within XIMPEL it can be designed in two different ways: fixed order or random order. Because the order of minigames does not matter for passing the final test, it is implemented as a randomized version. The player is informed before the test, that there will be eight games.

6.3 Navigation

Another aspect of the interaction graph in *Meet Rembrandt* are menus. The first part does not have one, so the player is forced to play the game from the beginning until the end. The two other parts give the possibility to choose a video to be watched by introducing a menu like in DVD movies.

In the second part of the game the menu appears only in the beginning. Players can choose which topic they want to learn about. It is not possible to go back to the menu during the game play, therefore all the topics that are placed in the storygraph before the chosen topic, are being skipped. In order to avoid such a situation, the third part of the game uses a menu that can be easily reached throughout the game (Fig.12). It is implemented as a permanent overlay, which allows users to jump between topics whenever they feel like. In *Meet Rembrandt*, such transitions cause some problems with scoring, but this could be solved in a technical way in the future. Such an approach gives players the feeling of control over topics, but might have a negative impact on the amount and the quality of knowledge, that players have gained. On the other hand, in opposite to a menu in the second part, it allows for playing all the segments, even if initially chosen topic was further in the storygraph.

The third part introduces also a timeline menu, giving the possibility to search the game not only within topics, but also periods (dates).

6.4 User Interaction

When creating *Meet Rembrandt*, more design considerations have been encountered, that apply to all three versions of the game. Some of the decisions have been made mostly because of XIMPEL's technical constraints. One of the questions was if the player should be forced to finish a minigame before they can proceed to the next video? In case of an educational game, where these small assignments check student's knowledge, this might be an important issue. Because of technical limitations, in *Meet Rembrandt* players are able to use the "next" button, but still, not finishing the game means that they get no points and will not reach the next level.

Another issue corresponds to the way in which video clips are being handled. When the player finishes watching a video with an overlay, where the decision has to be made, the clip starts from the beginning. It is implemented in such a way because of XIMPEL's limitations, and might be a good solution, as it allows students to focus on the video and make their decision only after they have seen all of it. It also gives the possibility to watch the video once again in case some information has been missed. For some people, however, it might be confusing.

Another issue connected with presentation of video clips is the question how should XIMPEL react to player's actions. When users need to make a decision by clicking on an overlay, XIMPEL may respond in two different ways: it can either wait for video to finish or it can move to the next video immediately. The latter is more interactive and instinctive, but it also means that players will not see the entire video and will skip some important information. In *Meet Rembrandt*, this option is implemented, because in new version 3.0, `<waitforvideocomplete>` tag is not supported.

Another important aspect is to make players aware of the amount of the material. It is not implemented in *Meet Rembrandt*, because of XIMPEL's limitations, but should be considered in the future development. Such improvements include having a time indication, so the player knows how long is each video (right now it is possible to check only when pushing "pause" button) and a progression bar, saying which percentage of all the material has been covered so far.

As mentioned before, an auditory information may contribute to learning, either directly, by allocating attention to educational content, or indirectly, when increased enjoyment motivates for studying. *Meet Rembrandt* uses audio in the second part only, giving the possibility to compare different versions.

6.5 The Effect of Minigames on the Storygraph

Extending XIMPEL with minigames affects how the playlists are being created. The storygraph needs to be designed in a way that supports students in their process of gaining knowledge in order to win minigames and reach the next level.

Some videos are obligatory to watch, because their content is necessary to answer the questions from a minigame. There are two ways to implement it in XIMPEL: either all the branches of a storygraph end at the same point, forcing players to watch such a video, or a notification is shown, which encourages to see a video, by informing players, that content from that clip is crucial to reach the next level. The first version is implemented in the third part of *Meet Rembrandt*, where all the branches end up at "Town Hall" subject. This video gives an important background information about "The Conspiracy of Claudius Civillis", which is necessary to win a minigame. The other version is used in the second part of the game, with a notification, that a tour through Rembrandt's house is required. Here, however, the player can decide if they are going to watch the video.

In most of the cases, minigames are based on the content of all the videos from the playlist. This means, that in order to get the best results, players should watch every video. It is possible to pass a minigame, when some of the clips have been skipped, but assuming that users make some mistakes, it is highly recommended to watch all of them. This requirement significantly affects the storygraph. A clear indication of what needs to be seen is important. If a video clip has more overlays, it can be confusing for students. This is why a storygraph should include some notifications, that explain which videos are necessary. Players should be aware, that they have missed essential information.

Using minigames as assignments before reaching the next level, requires introducing a new construct - a loop. Students need to win a minigame, otherwise they are sent back to watch necessary videos. Such loops force

players to study harder. It might be annoying for some of them to watch the same videos again, but repeating helps in memorizing the content. In *Meet Rembrandt*, the only implemented loop occurs in the second part, after the segment with Rembrandt's studio and his paintings. If the score in minigames is not high enough, the player is sent back to the studio. They are informed about their poor result and advised (forced) to watch the videos again, until their score is satisfactory. Such loops are meant to occur after each segment in all parts of the game, but are not implemented because of technical limitations.

The nature of minigames and in which cases they are being used, may also have an impact on the playlist. For example, in the second part of *Meet Rembrandt*, there are four games in a row after the topic about Rembrandt's paintings. Each minigame checks the player's knowledge from a different perspective (matching names, sorting dates, etc.). Paintings are essential in case of a game about Rembrandt, so this knowledge is checked more properly. Players might get bored or annoyed seeing a few games in a row and some of them would prefer to have one minigame after each of the paintings. However, because of the nature of minigames, it is more effective to have few paintings for each of them. In case of a memory game, it would even not be possible to create it with only one painting. Such requirements affect the way the storygraph is designed.

7. Feedback

I presented my game to eight people of different age, educational background, interests and computer skills. I showed them three parts of *Meet Rembrandt*, in order to compare different approaches in designing a serious game. Each version differs in terms of branching structure, the balance between minigames and storytelling, navigation and user interaction. To collect information I observed the players during the process. After each part I asked them to fill in a short survey. Such an informal experiment helped me to realize how other people perceive my game, what they expect and how react to different designs. In this chapter I present the results and my solutions to problematic issues.

In general everyone was happy with my game and considered it fun. I could see players getting engaged with every next step, in some cases even getting excited. They all felt encouraged to learn more and, with no exceptions, everyone would prefer to study using interactive video over regular books. The video material seemed to be appropriate and minigames were considered fun and effective at checking players' knowledge.

The structure of the game, even though it was changing, was clear to players in most of the cases. The first part was confusing for some of them, as they did not know what to expect. After having played for a couple of minutes, they understood the structure very well and were adapting easily to changes in the branching structure of the next parts of the game.

The result of skipping some videos was not clear to players in the first part. They realized it only after they have played (and in almost all cases - failed) the first minigame. From then on, they all started watching videos much more carefully.

To avoid situations such as described in two above examples, I decided to add an instruction screen in the beginning of the game, explaining the main idea of an interactive video game and overlays.

The balance of videos (educational material) and minigames (entertainment) was satisfying for players. Because of the nature of videos, some of the players considered an educational material even more appealing than minigames, so they did not mind if there were many clips. Those who liked minigames more than movies, still did not feel overloaded with the amount of videos. Some of them felt that there were a bit too many clips to watch within one screen in such parts of the game, like: studio, Night Watch or Conspiracy of Claudius Civillis. They did not think it was boring, but only

had some problems with memorizing all the information in order to win a minigame. When asked if they would have preferred binary structure, they all agreed that an existing multiway structure is more appealing and clear.

Once players have realized how important is to understand and remember information from videos, they started to watch them more carefully. Navigation buttons were used a lot - next/back arrows, as well as the pause, that allows to read text at a slower pace. The pause button, however, is a bit confusing, as it does not change to "play" when pushed. Navigation mechanism in XIMPEL should be fixed in the future.

Players did not care about the fact, that there is no indication of how long are videos or how much material has been covered. The videos are short enough not to make them bored. The majority was watching entire movies before even reading the branch question. Therefore, they were not annoyed by the fact, that they jump immediately after clicking on an overlay. Only one player experienced such a difficulty, but went back to the skipped video using navigation buttons. The fact that videos go in a loop after they have finished, was not annoying to anyone. Most of the players have even found it very helpful. They were watching each video as many times as they needed to understand and memorize presented information. As an effect, the results in minigames were much better.

In most cases minigames were considered not easy to win, which shows that knowledge is being checked properly. After having played the first part, hardly anyone said that they helped them memorize information. Minigames in part two and three received much better reviews. Players often did not have enough knowledge to win a minigame (because they have skipped some videos or did not remember), but they would still try. In case of sorting and matching game, such tryouts helped them to memorize information. Even though most of the players did not pass the loop in the second part and were sent back, they all have succeeded in their second attempt. They thought that such loops and minigames, dependent on total clicks, were helpful.

The idea of adding audio to boards with plain text met with different reactions. Most of the players were considering it helpful, but some of them thought it was harder to follow. They would prefer to only read a text and not being distracted by additional audio. It could be a good idea to extend XIMPEL with the possibility to mute audio in videos, as some of the people have better visual memory, and others - aural.

Some felt confused by the fact that after watching some videos, they were forced to read a plain text. The structure felt a bit inconsistent to them. It would be better if all videos were of a similar kind (now some of them are cut

from a documentary, and some are just textual), but in my case it was hard in realization (because of limited video material available in the Internet).

Overlays were easy to understand for players as long as they had a textual description. Overlays hidden in paintings were problematic. After my interviews I decided to add the information that also paintings can be chosen as an overlay.

The general trend was that players were quickly adapting to the structure and, if only sufficiently notified beforehand, they could follow the game easily. When they have realized the result of skipping some videos, they willingly watched all of them, tried harder and got better results, which encouraged them to learn even more. Some of them got to the point, where they could not wait to see what will happen later, and, therefore, got a bit distracted. All in all, reactions were very positive and promising. Most of the players declared that they want to visit museums with Rembrandt's paintings and read more about him, even though they did not have such a desire before playing the game.

8. Conclusions and Future Work

In my research I wanted to introduce minigames in XIMPEL, giving authors the possibility to include them in the playlist in a simple way. I achieved it by creating a minigame custom media type. I prepared four game templates, which can be easily configured even by inexperienced users. In my thesis I examined how these minigames can be used in interactive video, presenting an example of an interactive video game in XIMPEL with my customized game templates. I showed that minigames can be used in different cases to aid learning, by serving as an experience from which students learn, or checking players' knowledge.

I proved that extending XIMPEL with minigames affects how the playlists are being created. The storygraph needs to be designed in a way that supports students in their process of gaining knowledge in order to win minigames and reach the next level. Therefore, the branching structure needs to be adjusted, new constructs, for example a loop, need to be introduced and additional notifications are necessary.

Comparison of three different versions of *Meet Rembrandt* helped to draw a conclusion, that the design of the game may have an impact on what, and how, students learn. The branching structure, the balance between educational content and entertaining games, the navigation and user interaction influence the player's reception of the game, and, therefore, affect the learning process. Finding a proper balance between all these issues might not be an easy task, but can significantly improve the learning effect.

The feedback I have received on my game, shows that interactive video is a promising approach in education. Extending XIMPEL with minigames allows authors for creating powerful educational games in a very simple way. It also leaves open the possibility to improve the platform by adding more game templates in the future.

When designing my game, I encountered a few issues, which are worth discussing and improving in the future, but were out of the scope of this thesis.

XIMPEL 3.0 is the latest version of the platform, which introduces new functionality. Some issues, however, still need improvement. One of them is the navigation mechanism. XIMPEL Player uses navigation buttons, that allow moving to the previous or the next video, as well as pausing the clip. The pause button might cause confusion, as it does not change its icon to "play",

when pushed. Watching people playing *Meet Rembrandt*, I could observe them clicking "pause" a couple times, because they were thinking it did not work.

The way in which XIMPEL handles movies with overlays could also be improved. In the previous versions, the designer could decide whether XIMPEL Player should react immediately to user's actions. Adding a tag `<waitforvideocomplete>` would mean that when the player clicks on an overlay, they still need to wait for the video to finish. Such option is not supported in XIMPEL 3.0 anymore. Also, videos with overlays require a decision from the user. If one is not made, the game finishes at the end of the video, which is not a desired behavior. The only way to handle it, is adding a *repeat* attribute to the video in the playlist. This, however, implies the fact, that the video will go in a loop until the decision has been made, which can be confusing for the player. It would be good to give designers the possibility to choose if the video should be repeated, or just stopped, before one of the overlays is clicked.

Minigames are a completely new functionality in XIMPEL, that still requires development. When creating *Meet Rembrandt*, I encountered some problems, that should be fixed in the future. One of the questions was if the player should be forced to finish a minigame before they can proceed to the next video? Right now they can use navigation buttons and skip the minigame, but it should be possible for the designer to choose, which option they want to implement.

Another issue relates to the scoring mechanism in minigames. XIMPEL allows for using multiple score labels, that give more possibilities in profiling the user. This information can be used to create a more customized user experience, because there are more details available about choices that were made. Such score labels could be used in *Meet Rembrandt*, having a different label for each of the levels. Depending on the score in each of the levels, the player would either reach the next level, or be sent in the loop to the starting point. Unfortunately, minigames do not support this option. The points are always added to the main score at the end of the minigame. This should be fixed in the future, so that the game designer can determine whether his minigames use the main score or multiple labels.

Another problem with scoring occurs when the menu is introduced. Menus allow for jumping between different parts of the game, what causes a problem with the main score. Moving to a different section, the player still keeps his score from the previous parts. This could be solved, for example, by clearing the score every time, when the user jumps to another subject.

Except for improving already existing functionality, there are few issues, that came up during the design process of *Meet Rembrandt*. These are features,

that have yet not been supported in XIMPEL. One of the ideas is to give the player the possibility to check how much material needs to be covered. In order to do so, a progression bar could be introduced. It would show the percentage of the material that has been already learned and how much is left. Also a time indication is a desired functionality. In XIMPEL 3.0, players can not see how long is the video they are watching.

Another nice functionality would be to give users the possibility to mute audio in the videos, because some of them feel distracted when reading and listening at the same time.

The last issue that came up while designing *Meet Rembrandt*, is the possibility to disable overlays. At the end of the game, there is a final test, which covers information from all three parts of the demo. It consists of eight different minigames, that check players' knowledge from different perspectives. In XIMPEL, this could be implemented by creating one screen with eight overlays, that allows players to choose the order of playing. After each minigame, the player would be sent back to the starting screen and, if the last game had been passed, that overlay would be disabled. XIMPEL does not support such functionality, but it should be improved in the future.

References:

1. *Surrounded by Science: Learning Science in Informal Environments*, Marilyn Fenichel and Heidi A. Schweingruber; National Research Council, 2010. Available online at www.nap.edu.
2. *How People Learn: Brain, Mind, Experience, and School: Expanded Edition*, Committee on Developments in the Science of Learning with additional material from the Committee on Learning Research and Educational Practice, National Research Council, 2000. Available online at www.nap.edu.
3. *Games for science and engineering education*, Merrilea J. Mayo, Communications of the ACM, July 2007, Vol.50, No.7.
4. *Learning theories, A to Z*, David C. Leonard, Greenwood, 2002.
5. *Learning Science Through Computer Games and Simulations*, Committee on Science Learning: Computer Games, Simulations, and Education; Margaret A. Honey and Margaret Hilton, Editors; National Research Council, 2011. Available online at www.nap.edu.
6. *Serious games + computer science = Serious CS*, Katrin Becker and J. R. Parker, Journal of Computing Sciences in Colleges, Volum 23 Issue 2, December 2007.
7. *Math game(s) - an alternative (approach) to teaching math?*, Anton Eliens and Zsofia Ruttkay, 2009.
8. *Essential ActionScript 3.0*, Colin Moock, O'REILLY, 2007.
9. *Foundation ActionScript 3.0 with Flash CS3 and Flex*, Steve Webster, Todd Yard, Sean McSharry, 2008.
10. *Exploring Models of Interactivity from Multiple Research Traditions: Users, Documents, And Systems*, Sally J. McMillan, Handbook of New Media (p. 162-182), London, 2002.
11. *How to Build Serious Games*, Henry Kelly, Kay Howell, Eitan Glinert, Loring Holding, CHRis Swain, Adam Burrowbridge and Michelle Roper, Communications of the ACM, July 2007, Vol.50, No.7.
12. *Tradition and Technology in the Classroom*, *Little Planet Literacy Series*, Vanderbilt University, Volume 2, Issue 2, Spring 1997:
http://www.vanderbilt.edu/News/research/ravs97/ravs97_3.html

13. *Adobe Flex*: <http://www.adobe.com/products/flex.html>
14. *XIMPEL*: ximpel.net
15. *The Voyage of the Mimi*: http://en.wikipedia.org/wiki/The_Voyage_of_the_Mimi, accessed at 14 October 2011.
16. *Experiential learning*: http://en.wikipedia.org/wiki/Experiential_learning, accessed at 14 October 2011.
17. *Self-efficacy*: <http://en.wikipedia.org/wiki/Self-efficacy>, accessed at 14 October 2011.
18. *Goal setting*: http://en.wikipedia.org/wiki/Goal_setting, accessed at 14 October 2011.

Meet Rembrandt - Resources:

The game is available at:

Part One: <http://www.few.vu.nl/~cbk350/rembrandt/part1/ximpelApp-online.swf>

Part Two: <http://www.few.vu.nl/~cbk350/rembrandt/part2/ximpelApp-online.swf>

Part Three: <http://www.few.vu.nl/~cbk350/rembrandt/part3/ximpelApp-online.swf>

The playlists are available at:

Part One: <http://www.few.vu.nl/~cbk350/rembrandt/part1/playlists/fast.xml>

Part Two: <http://www.few.vu.nl/~cbk350/rembrandt/part2/playlists/fast.xml>

Part Three: <http://www.few.vu.nl/~cbk350/rembrandt/part3/playlists/fast.xml>

Appendix A

1. The Code of Minigame Media Type - minigame.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:HBox xmlns:mx="http://www.adobe.com/2006/mxml"
  cornerRadius="2" width="800" height="600" x="0" y="0" horizontalAlign="center"
  initialize="init()" implements="net.ximpel.media.IMediaType">

  <mx:Script>
    <![CDATA[

      import net.ximpel.events.MediaEvent;
      import flash.net.LocalConnection;

      private const MINIGAME_EXTENSION:String = ".swf";

      private var _incoming_lc:LocalConnection;

      private function init():void {
        initLocalConnection();
      }

      private function onUnload(event:Event):void {
        //trace("minigame unloaded");
      }

      private function initLocalConnection():void {
        _incoming_lc = new LocalConnection();
        _incoming_lc.connect("minigame_status");
        _incoming_lc.client = this;
      }

      public function onMinigameComplete(status:String,score:int):void {
        stopMedia();
        dispatchEvent(new MediaScoreEvent
          (MediaScoreEvent.SCORE_RESULT,false,false,score));
        dispatchEvent(new MediaEvent(MediaEvent.COMPLETE,
          false, false, typeName));
      }

      public function get typeName():String {
        return "minigame";
      }

      public function playMedia(mediaData:XML):Boolean {
```

```

        var file:String = mediaData.@file;
        mySWFLoader.source = file;
        return true;
    }

    public function togglePauseMedia():void {
        //do nothing
    }

    public function stopMedia():Boolean {
        mySWFLoader.source = "";
        return true;
    }
    ]]>
</mx:Script>

    <mx:SWFLoader id="mySWFLoader" bottom="0" unload="onUnload(event)"/>
</mx:HBox>

```

2. Templates

2.1 Memory Game

```

<?xml version="1.0"?>
<!-- Minigame Memory -->
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="cardImageService.send()">

    <mx:HTTPService
        id="cardImageService"
        url="assets/memory/memorycards.xml"
        resultFormat="e4x"
        result="cardImageResultHandler(event);"
        fault="cardImageFaultHandler(event);"/>

    <mx:Script>
        <![CDATA[

            import mx.rpc.events.FaultEvent;
            import mx.rpc.events.ResultEvent;
            import mx.controls.*;
            import mx.events.*;
            import flash.utils.*;
            import flash.net.LocalConnection;

```

```
private var outgoing_lc:LocalConnection = new LocalConnection();
```

```
[Bindable]
```

```
private var gameArray:Array;  
private var oneCardFlipped:Boolean;  
private var currentCard:String;  
private var previousCard:String;  
private var currentCardIndex:int;  
private var previousCardIndex:int;  
private var cardPairsTaken:int;  
private var timerActive:Boolean;  
private var genericCardSide:String;  
private var assetsPath:String;  
private var playerScore:int;  
private var totalClicks:int;  
private var cardImageFeed:XML;  
private var intervalID:uint;  
private var welcome:String;
```

```
private function initMemoryGame():void {
```

```
    playerScore = 0;  
    cardPairsTaken = 0;  
    oneCardFlipped = false;  
    currentCard = "";  
    previousCard = "";  
    currentCardIndex = -1;  
    previousCardIndex = -1;  
    timerActive = false;  
    setScore();  
    totalClicks = 0;  
    setTotalClicks();  
    setPairsTaken();  
    info.text=welcome;  
    fillArray();
```

```
}
```

```
private function fillArray():void {
```

```
    gameArray = new Array(cardImageFeed.card.length()*2);  
    //We need two of each pictures
```

```
    for (var i:int = 0; i < gameArray.length; i++ )
```

```
    {  
        gameArray[i] = assetsPath + cardImageFeed.card[0].url;  
    }
```

```
    var random:int;
```

```
    var j:int;
```

```

for (var k:int = 1; k < cardImageFeed.card.length(); k++){
    j = 0;
    while (j < 2){
        random = Math.random()*gameArray.length;
        if (gameArray[random] == assetsPath +
            cardImageFeed.card[0].url){
            gameArray[random] = assetsPath +
                cardImageFeed.card[k].url;
            j++;
        }
    }
}

private function reveal(cardIndex:int):void {
    if (timerActive) return;

    currentCardIndex = cardIndex;
    currentCard = card[currentCardIndex].source =
        gameArray[currentCardIndex];

    if (currentCardIndex == previousCardIndex) return;
        //Clicking on the already turned card

        totalClicks++;
        setTotalClicks();
        setScore();

        if (!oneCardFlipped){
            previousCardIndex = currentCardIndex;
            previousCard = currentCard;
            oneCardFlipped = true;
        }else {
            var timer:Timer = new Timer(800,1);
            timer.addEventListener(TimerEvent.TIMER,
                dispatchComplete);
            timerActive = true;
            timer.start();
        }
}

private function dispatchComplete(event:TimerEvent):void {
    if (!timerActive) return;

    if (currentCard == previousCard){
        card[previousCardIndex].source = "";
    }
}

```

```

        //Remove cards from play
        card[currentCardIndex].source = "";
        cardPairsTaken++;
        setPairsTaken();
        setScore();
    }else {
        card[previousCardIndex].source =
        card[currentCardIndex].source = genericCardSide;
    }

    previousCardIndex = -1;
    previousCard = "";
    oneCardFlipped = false;
    timerActive = false;

    if (cardPairsTaken*2 == gameArray.length) {
        info.text="Congratulations! You found all pairs!";
        intervalID = setInterval (sendEndGameMessage, 1000);
    }
}

private function sendEndGameMessage():void {

    clearInterval(intervalID);
    outgoing_lc.send("minigame_status", "onMinigameComplete",
                    "minigame_done",playerScore);
}

private function setGenericCardSide():String {
    return genericCardSide;
}

private function setScore():void {
    playerScore = Math.floor((20 * cardPairsTaken) / totalClicks);
    scoreLabel.text = "Score: " + playerScore;
}

private function setTotalClicks():void {
    totalClicksLabel.text = "Total clicks: " + totalClicks;
}

private function setPairsTaken():void {
    pairsTakenLabel.text = "Pairs taken: " + cardPairsTaken;
}

// Result handler - gets called after feed is loaded
private function cardImageResultHandler(event:ResultEvent):void {

```

```

        cardImageFeed = event.result as XML;

        var pattern:RegExp = /.*\//;
        var results:Array = pattern.exec(cardImageService.url);
        results != null ? assetsPath = results[0] : assetsPath = "";
        genericCardSide = assetsPath +
            cardImageFeed.genericCardSide[0].url;

        welcome = cardImageFeed.welcometxt[0].txt;
        initMemoryGame();
    }

    // Fault handler - displays the error
    private function cardImageFaultHandler(event:FaultEvent):void {
    Alert.show(event.fault.message, "Could not load cardImage feed");
    }
    ]]>
</mx:Script>

<mx:Panel id="titlePanel" title="Memory Game" width="75%" height="75%"
    paddingTop="10" paddingLeft="10" paddingRight="10" paddingBottom="10">

<mx:VBox width="100%">
    <mx:Text id="info" width="100%" paddingTop="20"/>

    <mx:HBox borderStyle="solid">
        <mx:Label id="pairsTakenLabel" height="20" width="100" text=""/>
        <mx:Label id="totalClicksLabel" height="20" width="100" text=""/>
        <mx:Label id="scoreLabel" height="20" width="100" text=""/>
    </mx:HBox>

    <mx:Tile direction="horizontal" borderStyle="inset"
        horizontalGap="10" verticalGap="15"
        paddingLeft="10" paddingTop="10" paddingBottom="10" paddingRight="10">
        <mx:Repeater id="rp" dataProvider="{gameArray}">
            <mx:Image id="card" width="100" height="100"
                source="{setGenericCardSide()}"

                click="reveal(event.currentTarget.repeaterIndices)"
                horizontalAlign="center" verticalAlign="middle"/>
        </mx:Repeater>
    </mx:Tile>

</mx:VBox>

</mx:Panel>
</mx:Application>

```

2.2 Sorting Game

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application
  xmlns:mx="http://www.adobe.com/2006/mxml"
  initialize="cardImageService.send()">

  <mx:HTTPService
    id="cardImageService"
    url="assets/sorting/sortingImages.xml"
    resultFormat="e4x"
    result="cardImageService_resultHandler(event)"
    fault="cardImageService_faultHandler(event)"/>

  <mx:Script>
    <![CDATA[
      import flash.net.LocalConnection;

      import mx.collections.*;
      import mx.controls.*;
      import mx.core.DragSource;
      import mx.events.DragEvent;
      import mx.logging.*;
      import mx.managers.DragManager;
      import mx.rpc.events.FaultEvent;
      import mx.rpc.events.ResultEvent;

      private var outgoing_lc:LocalConnection = new LocalConnection();

      private var imageFeed:XML;
      private var assetsPath:String;
      [Bindable]
      private var gameArray:Array;
      [Bindable]
      private var urlAC:ArrayCollection = new ArrayCollection();
      private var playerScore:int;
      private var priority:int;
      private var expectedPriority:int;
      private var intervalID:uint;
      private var totalClicks:int;
      private var sortedImages:int;
      private var welcome:String;

      private function initSortingGame():void
      {
        playerScore = 0;
      }
    ]]>
  </mx:Script>
</mx:Application>
```

```

totalClicks=0;
sortedImages=0;
setScore();
setTotalClicks();
setSortedImages();
priority= 0;
expectedPriority=1;
info.text= welcome;
fillArray();
}

private function fillArray():void
{
    gameArray = new Array(imageFeed.image.length());

    for ( var i:int = 0; i < gameArray.length; i++ )
    {
        gameArray[i] = assetsPath + imageFeed.image[0].url;
    }

    var random:int;
    var j:int;
    var k:int;

    for (k = 1; k < imageFeed.image.length(); k++)
    {
        j=0;
        while (j < 1)
        {
            random = Math.random()*gameArray.length;
            if (gameArray[random] == assetsPath +
                imageFeed.image[0].url)
            {
                gameArray[random] = assetsPath +
                    imageFeed.image[k].url;
                j++;
            }
        }
    }

    urlAC= new ArrayCollection(gameArray);
}

```

```

private function checkPriority (myPath:String):void
{
    for (var m:int=0; m < imageFeed.image.length(); m++)
    {
        if (myPath == assetsPath + imageFeed.image[m].url)
        {
            priority = imageFeed.image[m].priority;
        }
    }
}

```

```

private function initiateDrag(event:MouseEvent,path:String):void
{
    checkPriority(path);

    var dragInitiator:Image = Image(event.currentTarget);
    var datasource:DragSource = new DragSource();
    datasource.addData(priority, "priority");
    datasource.addData(path, "path");

    var dragProxy:Image = new Image();
    dragProxy.source = event.currentTarget.source;
    dragProxy.width = 120 ;
    dragProxy.height = 120 ;

```

```

DragManager.doDrag(dragInitiator, datasource, event, dragProxy);
}

```

```

private function dragEnterHandler(event:DragEvent):void
{
    var dropTarget:HBox =event.currentTarget as HBox;

    DragManager.acceptDragDrop(dropTarget);
}

```

```

private function dragDropHandler(event:DragEvent):void
{
    if (event.dragSource.hasFormat("priority") &&
        event.dragSource.dataForFormat("priority")==expectedPriority)
    {
        Image(event.dragInitiator).source="";
        expectedPriority++;
        totalClicks++;
    }
}

```

```

        sortedImages++;
        setTotalClicks();
        setSortedImages();
        setScore();
        sortImage.source=Image(event.dragInitiator).source;
        var sortImagePath:String =
        event.dragSource.dataForFormat('path') as String;
        sortImage.source=sortImagePath;
    }

    else if (event.dragSource.hasFormat("priority") &&
    event.dragSource.dataForFormat("priority"!
==expectedPriority)
    {
        totalClicks++;
        setTotalClicks();
        setScore();
    }

    if (sortedImages == gameArray.length)
    {
        info.text="Congratulations!";
        intervalID = setInterval (sendEndGameMessage, 1000);
    }
}

private function sendEndGameMessage():void
{
    clearInterval(intervalID);
    outgoing_lc.send("minigame_status", "onMinigameComplete",
    "minigame_done",playerScore);
}

private function setScore():void
{
    playerScore = Math.floor((8 * sortedImages) / totalClicks);
    scoreLabel.text = "Score: " + playerScore;
}

private function setTotalClicks():void {
    totalClicksLabel.text = "Total clicks: " + totalClicks;
}

private function setSortedImages():void {
    sortedImagesLabel.text = "Sorted images: " + sortedImages;
}

```

```

    }

    protected function
    cardImageService_resultHandler(event:ResultEvent):void
    {
        imageFeed = event.result as XML;

        var pattern:RegExp = /.*\//;
        var results:Array = pattern.exec(cardImageService.url);
        results != null ? assetsPath = results[0] : assetsPath = "";

        welcome = imageFeed.welcometxt[0].txt;

        initSortingGame();
    }

    protected function
    cardImageService_faultHandler(event:FaultEvent):void
    {
        Alert.show(event.fault.message, "Could not load image feed");
    }

    ]]>
</mx:Script>

```

```

<mx:Panel id="titlePanel" title="Sorting Game" width="100%" height="100%"
layout="absolute" paddingTop="10" paddingBottom="10"
paddingLeft="10" paddingRight="10">

```

```

    <mx:VBox x="10" y="10" width="100%" horizontalAlign="center"
verticalAlign="middle" paddingBottom="30">

```

```

    <mx:Text id="info" paddingTop="20" />

```

```

    <mx:HBox borderStyle="solid">

```

```

    <mx:Label id="sortedImagesLabel" height="20" width="120" text="" />

```

```

    <mx:Label id="totalClicksLabel" height="20" width="100" text="" />

```

```

    <mx:Label id="scoreLabel" height="20" width="100" text="" />

```

```

    </mx:HBox>

```

```

    <mx:HBox width="100%" height="100%" horizontalAlign="center"

```

```

verticalAlign="middle" horizontalGap="40"
paddingBottom="20" paddingTop="20" paddingLeft="20" paddingRight="20">
    <mx:Repeater id="rp" dataProvider="{urlAC}">
        <mx:Image width="120" height="120" source="{rp.currentItem}"

mouseMove="initiateDrag(event,event.currentTarget.getRepeaterItem())"/>
    </mx:Repeater>
</mx:HBox>
<mx:HBox width="200" height="200" backgroundColor="#000000"
horizontalAlign="center" verticalAlign="middle"
dragEnter="dragEnterHandler(event)"
dragDrop="dragDropHandler(event)">
    <mx:Image id="sortImage"/>
</mx:HBox>

</mx:VBox>

</mx:Panel>

</mx:Application>

```

2.3 Matching Game

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application
xmlns:mx="http://www.adobe.com/2006/mxml"
initialize="cardImageService.send()">

<mx:HTTPService
id="cardImageService"
url="assets/matching/matchingImages.xml"
resultFormat="e4x"
result="cardImageService_resultHandler(event)"
fault="cardImageService_faultHandler(event)"/>

<mx:Script>
<![CDATA[
import flash.net.LocalConnection;

import mx.collections.*;
import mx.controls.*;
import mx.core.DragSource;
import mx.events.DragEvent;
import mx.logging.*;
import mx.managers.DragManager;

```

```

import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;

private var outgoing_lc:LocalConnection = new LocalConnection();

private var imageFeed:XML;
private var assetsPath:String;
[Bindable]
private var gameArray:Array;
[Bindable]
private var gameArrayMatch:Array;
[Bindable]
private var urlAC:ArrayCollection = new ArrayCollection();
[Bindable]
private var matchAC:ArrayCollection = new ArrayCollection();
private var playerScore:int;
private var matchTarget:String;
private var intervalID:uint;
private var totalClicks:int;
private var pairsTaken:int;
private var welcome:String;

private function initMatchingGame():void
{
    playerScore = 0;
    totalClicks=0;
    pairsTaken=0;
    setScore();
    setTotalClicks();
    setPairsTaken();
    matchTarget = "";
    info.text=welcome;
    fillArrays();
}

private function fillArrays():void
{
    gameArray = new Array(imageFeed.image.length());
    gameArrayMatch = new Array(imageFeed.image.length());

    for ( var i:int = 0; i < gameArray.length; i++ )
    {
        gameArray[i] = assetsPath + imageFeed.image[0].url;
        gameArrayMatch[i] = assetsPath +
            imageFeed.image[0].match;
    }
}

```

```

    }

    var random:int;
    var j:int;
    var k:int;

    for (k = 1; k < imageFeed.image.length(); k++)
    {
        j=0;
        while (j < 1)
        {
            random = Math.random()*gameArray.length;
            if (gameArray[random] == assetsPath +
                imageFeed.image[0].url)
            {
                gameArray[random] = assetsPath +
                    imageFeed.image[k].url;
                j++;
            }
        }
    }

    for (k = 1; k < imageFeed.image.length(); k++)
    {
        j=0;
        while (j < 1)
        {
            random = Math.random()*gameArrayMatch.length;
            if (gameArrayMatch[random] == assetsPath +
                imageFeed.image[0].match)
            {
                gameArrayMatch[random] = assetsPath +
                    imageFeed.image[k].match;
                j++;
            }
        }
    }

    fillAC();
}

private function fillAC():void
{
    urlAC= new ArrayCollection(gameArray);
}

```

```

        matchAC= new ArrayCollection(gameArrayMatch);
    }

private function findMatch(myPath:String):void
{
    for (var m:int=0; m < imageFeed.image.length(); m++)
    {
        if (myPath == assetsPath + imageFeed.image[m].url)
        {
            matchTarget= assetsPath + imageFeed.image[m].match;
        }
    }
}

private function initiateDrag(event:MouseEvent,path:String):void
{
    findMatch(path);

    var dragInitiator:Image = Image(event.currentTarget);
    var datasource:DragSource = new DragSource();
    datasource.addData(matchTarget, "match");

    var dragProxy:Image = new Image();
    dragProxy.source = event.currentTarget.source;
    dragProxy.width = 100 ;
    dragProxy.height = 100 ;

    DragManager.doDrag(dragInitiator, datasource, event, dragProxy);

}

private function dragEnterHandler(event:DragEvent):void
{
    var dropTarget:Image =event.currentTarget as Image;

    DragManager.acceptDragDrop(dropTarget);

}

private function dragDropHandler(event:DragEvent,
    pathMatch:String):void
{
    if (event.dragSource.hasFormat("match") &&
        event.dragSource.dataForFormat("match")==pathMatch)
    {

```

```

        event.currentTarget.source="";
        Image(event.dragInitiator).source="";
        pairsTaken++;
        totalClicks++;
        setPairsTaken();
        setTotalClicks();
        setScore();
    }

    else if (event.dragSource.hasFormat("match") &&
event.dragSource.dataForFormat("match")!==pathMatch)
    {
        totalClicks++;
        setTotalClicks();
        setScore();
    }

    if (pairsTaken == gameArray.length)
    {
        info.text="Congratulations! You found all pairs!";
        intervalID = setInterval (sendEndGameMessage, 1000);
    }
}

private function sendEndGameMessage():void
{
    clearInterval(intervalID);
    outgoing_lc.send("minigame_status", "onMinigameComplete",
"minigame_done",playerScore);
}

private function setScore():void
{
    playerScore = Math.floor((8 * pairsTaken) / totalClicks);
    scoreLabel.text = "Score: " + playerScore;
}

private function setTotalClicks():void {
    totalClicksLabel.text = "Total clicks: " + totalClicks;
}

private function setPairsTaken():void {
    pairsTakenLabel.text = "Pairs taken: " + pairsTaken;
}

```

```

protected function
cardImageService_resultHandler(event:ResultEvent):void
{
    imageFeed = event.result as XML;

    var pattern:RegExp = /.*\//;
    var results:Array = pattern.exec(cardImageService.url);
    results != null ? assetsPath = results[0] : assetsPath = "";

    welcome = imageFeed.welcometxt[0].txt;

    initMatchingGame();
}

```

```

protected function
cardImageService_faultHandler(event:FaultEvent):void
{
    Alert.show(event.fault.message, "Could not load image feed");
}

```

```

]]>
</mx:Script>

```

```

<mx:Panel id="titlePanel" title="Matching Game" width="100%" height="100%"
layout="absolute" paddingTop="10" paddingBottom="10" paddingLeft="10"
paddingRight="10">

```

```

<mx:VBox x="10" y="10" width="100%">

```

```

<mx:Text id="info" paddingTop="20" />

```

```

<mx:HBox borderStyle="solid">

```

```

<mx:Label id="pairsTakenLabel" height="20" width="100" text="" />

```

```

<mx:Label id="totalClicksLabel" height="20" width="100" text="" />

```

```

<mx:Label id="scoreLabel" height="20" width="100" text="" />

```

```

</mx:HBox>

```

```

<mx:HBox width="100%" height="100%" horizontalAlign="center"
verticalAlign="middle" horizontalGap="40"
paddingBottom="20" paddingTop="20" paddingLeft="20" paddingRight="20">
    <mx:Repeater id="rp" dataProvider="{urlAC}">
        <mx:Image width="100" height="100" source="{rp.currentItem}"

```

```

        mouseMove="initiateDrag(event,event.currentTarget.getRepeaterItem())"/>
        </mx:Repeater>
    </mx:HBox>
    <mx:HBox width="100%" height="100%" horizontalAlign="center"
verticalAlign="middle" horizontalGap="40"
paddingBottom="20" paddingTop="20" paddingLeft="20" paddingRight="20">
        <mx:Repeater id="rp1" dataProvider="{matchAC}">
            <mx:Image width="100" height="100"
source="{rp1.currentItem}"
            dragEnter="dragEnterHandler(event)"
            dragDrop="dragDropHandler(event,
event.currentTarget.getRepeaterItem())"/>
        </mx:Repeater>
    </mx:HBox>
</mx:VBox>

</mx:Panel>

</mx:Application>

```

2.4 Questionnaire

2.4.1 questionnaire.mxml

```

<?xml version="1.0" encoding="utf-8"?>
<mx:Application
    xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:local="*"
    initialize="questionService.send()">

    <mx:HTTPService
        id="questionService"
        url="assets/questionnaire/questions.xml"
        resultFormat="e4x"
        result="questionService_resultHandler(event)"
        fault="questionService_faultHandler(event)"/>

    <mx:Script>
        <![CDATA[
            import flash.net.LocalConnection;

            import mx.collections.*;
            import mx.controls.*;
            import mx.logging.*;
            import mx.rpc.events.FaultEvent;

```

```

import mx.rpc.events.ResultEvent;

private var outgoing_lc:LocalConnection = new LocalConnection();

private var xml:XML;
[Bindable]
private var questions:ArrayCollection;
private var intervalID:uint;
[Bindable]
private var playerScore:int;
private var welcome:String;
private var numQ:int;

private function initTest():void
{
    playerScore=0;
    info.text= welcome;
    end.text=" ";

    questions = new ArrayCollection();
    for each (var node : XML in xml.question)
    {
        questions.addItem(new Question(node));
    }

    numQ = questions.length;
}

private function sendEndGameMessage():void
{
    clearInterval(intervalID);
    outgoing_lc.send("minigame_status", "onMinigameComplete",
"minigame_done", playerScore);
}

protected function
questionService_resultHandler(event:ResultEvent):void
{
    xml = event.result as XML;
    welcome = xml.welcometxt[0].txt;
    initTest();
}

protected function
questionService_faultHandler(event:FaultEvent):void

```

```

    {
    Alert.show(event.fault.message, "Could not load question feed");
    }

    protected function
    submitButton_clickHandler(event:MouseEvent):void
    {
        end.text="You gave correct answer to " + playerScore + " out of "
            + numQ + " questions!";
        intervalID = setInterval (sendEndGameMessage, 2000);
    }

    ]]>
</mx:Script>

```

```

<mx:Panel id="titlePanel" title="Questionnaire" width="100%" height="100%"
paddingTop="10" paddingBottom="10" paddingLeft="10" paddingRight="10">

    <mx:VBox x="10" y="10" width="100%" horizontalAlign="center"
verticalAlign="middle" paddingBottom="20">

        <mx:Text id="info" paddingTop="20" paddingBottom="20" />

        <mx:Repeater dataProvider="{questions}" id="questionRepeater">
            <mx:Label text="{questionRepeater.currentItem.question}" />
            <local:AnswerGroup
                answers="{questionRepeater.currentItem.answers}"

                rightAnswerEvent="playerScore++"
                wrongAnswerEvent="playerScore--" />
        </mx:Repeater>

        <mx:Button label="Submit" id="submitButton"
click="submitButton_clickHandler(event)"/>

        <mx:Text id="end" paddingTop="20" paddingBottom="20" />

    </mx:VBox>

</mx:Panel>

</mx:Application>

```

2.4.2 Question.as

```
package
{
    import mx.collections.ArrayCollection;

    public class Question {

        public var question : String;
        public var answers : ArrayCollection = new ArrayCollection();

        public function Question(node:XML)
        {
            question = node.q.text();
            for each (var answer:XML in node.answer)
            {
                answers.addItem(new Answer(answer));
            }
        }
    }
}
```

2.4.3 AnswerGroup.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:VBox xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Metadata>
        [Event(name="rightAnswerEvent", type="AnswerEvent")]
        [Event(name="wrongAnswerEvent", type="AnswerEvent")]
    </mx:Metadata>
    <mx:Script>
        <![CDATA[
            import mx.collections.ArrayCollection;

            [Bindable]
            public var answers:ArrayCollection;
            public var previous:String;
```

```

protected function answerGroup_changeHandler(event:Event):void
    {
        if (answerGroup.selectedValue.correct == "true")
        {
            previous="true";
            dispatchEvent(new
                AnswerEvent(AnswerEvent.RIGHT_ANSWER_EVENT));
        }

        else if (answerGroup.selectedValue.correct == "false")
        {
            if (previous=="true")
            {
                previous="false";
                dispatchEvent(new
                    AnswerEvent(AnswerEvent.WRONG_ANSWER_EVENT));
            }
        }
    }
    ]]>
</mx:Script>

<mx:RadioButtonGroup change="answerGroup_changeHandler(event)"
id="answerGroup" />
<mx:Repeater dataProvider="{answers}" id="answersRepeater">
    <mx:RadioButton group="{answerGroup}"
        label="{answersRepeater.currentItem.text}"
        value="{answersRepeater.currentItem}" />
</mx:Repeater>
</mx:VBox>

```

2.4.4 Answer.as

```

package
{
    public class Answer {
        public var text : String;
        public var correct : String;

        public function Answer(node:XML)
        {
            text = node.toString();
            correct = String(node.@value);
        }
    }
}

```

2.4.5 AnswerEvent.as

```
package
{
    import flash.events.Event;

    public class AnswerEvent extends Event
    {
        public static const RIGHT_ANSWER_EVENT:String = "rightAnswerEvent";
        public static const WRONG_ANSWER_EVENT:String =
            "wrongAnswerEvent";

        public function AnswerEvent(type:String)
        {
            super(type);
        }
    }
}
```

Appendix B

Tutorial: MINIGAMES in XIMPEL

1. Creating a Minigame

Requires programming skills in Flex (Action Script and MXML)

1.1 Making Minigames Compatible with XIMPEL Player

Depending on the version of Action Script we use, one of these pieces needs to be added to minigame's code:

[ActionScript 2]

```
var outgoing_lc:LocalConnection = new LocalConnection();  
var status = "minigame_done";  
var score:int;  
minigame.sendEndGameMessage = function(Void):Void  
{  
    outgoing_lc.send("minigame_status", "onMinigameComplete",  
        status,score); }
```

[ActionScript 3]

```
import flash.net.LocalConnection;  
  
private var outgoing_lc:LocalConnection = new LocalConnection();  
private function sendEndGameMessage():void  
{  
    private var status:String = "minigame_done";  
    private var score:int;  
    outgoing_lc.send("minigame_status", "onMinigameComplete", status, score);  
}
```

Via a local connection with the name *"minigame_status"*, the function *onMinigameComplete* is called with 2 arguments: *status* and *score*. If *status* *"minigame_done"* is sent, XIMPEL understands, that the minigame has been

completed and it sends the score to the main application (this way general score can be updated with results from each minigame). The way the score is counted, should be indicated in minigame's code.

Once MXML file is ready, we need to create an SWF file, which can be easily done using Flex SDK.

1.2 How Does It Work?

One of XIMPEL's features is the possibility to register custom media types. To program a custom media type you need to create a class that implements the *IMediaType* interface. In order to use minigames in XIMPEL Player, *Minigame* class has been created. Every time a new minigame is added to the playlist, a new object is created with its local connection, that enables sending the status and score of each minigame to the main application.

To make this custom media type accessible in XIMPEL Player, it has been registered in the main application:

```
var minigame:Minigame = new Minigame();
minigame.addEventListener(MediaScoreEvent.SCORE_RESULT,updateMediaScore);
myXimpelPlayer.registerMediaType(minigame);
```

2. Templates

Does not require programming skills

A couple of minigame templates is available for users. Customizing them does not require any programming skills, so that everyone can easily create their own games. The templates use dynamic XML files, which include paths to the files and other variables. In folder named '*assets*', you can find directories for each type of a minigame. This is where the files are being stored.

2.1 Memory Game

Does not require programming skills

In *assets/memory* you can find *memorycards.xml*. Do not change the name of the file! An exemplary code looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<memoryCards>

    <welcometxt>
        <txt>Find all pairs. Less clicks = higher score!</txt>
    </welcometxt>

    <genericCardSide>
        <url>rama.jpg</url>
    </genericCardSide>
    <card>
        <url>memanatomy.jpg</url>
    </card>
    <card>
        <url>memsamson.jpg</url>
    </card>
    <card>
        <url>memsaskiared.jpg</url>
    </card>
    <card>
        <url>memruts.jpg</url>
    </card>
    <card>
        <url>memmaria.jpg</url>
    </card>

</memoryCards>
```

In order to create a memory game with your pictures, you need to change the names of the files within `<url>` tags. `genericCardSide` is the picture that will be displayed if the card is not flipped.

Within `<welcometxt><txt>` tag you can change text with instructions for your memory game.

Requires some programming skills

It is also possible to adjust memory game by changing the main code in `memory.mxml` file. Such adjustments include:

- how the score is counted - in the original version, total clicks influence the score (less clicks = higher score); the formula used to count the score: `Math.floor((20 * cardPairsTaken) / totalClicks)`; this can be changed in the code;
- changing the size of images; in the original version they are displayed as 100x100px.

2.2 Matching Game

Does not require programming skills

In `assets/matching` you can find `matchingImages.xml`. Do not change the name of the file! An exemplary code looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>
  <matchingImages>

  <welcometxt>
  <txt>Match Rembrandt's paintings with their names!</txt>
  </welcometxt>

  <image>
    <url>matchmaria.jpg</url>
    <match>matchmaria1.jpg</match>
  </image>

  <image>
    <url>matchsamson.jpg</url>
    <match>matchsamson1.jpg</match>
  </image>
```

```
<image>
  <url>matchanatomy.jpg</url>
  <match>matchanatomy1.jpg</match>
</image>
```

```
<image>
  <url>matchsaskiared.jpg</url>
  <match>matchsaskiared1.jpg</match>
</image>
```

```
<image>
  <url>matchruts.jpg</url>
  <match>matchruts1.jpg</match>
</image>
```

```
</matchingImages>
```

In order to create a matching game with your pictures, you need to change the names of the files within `<url>` and `<match>` tags. Make sure, that the matching images are within one `<image>` tag! Within `<welcometxt><txt>` tag you can change text with instructions for your matching game.

Requires some programming skills

It is also possible to adjust matching game by changing the main code in `matching.mxml` file. Such adjustments include:

- how the score is counted - in the original version, total clicks influence the score (less clicks = higher score); the formula used to count the score: $\text{Math.floor}((8 * \text{pairsTaken}) / \text{totalClicks})$; this can be changed in the code;
- changing the size of images; in the original version they are displayed as 100x100px. The semi-transparent image displayed when dragging is called a *drag proxy*. When changing the size of the image, you might also want to change the size of the proxy in `initiateDrag` function.

2.3 Sorting Game

Does not require programming skills

In *assets/sorting* you can find *sortpaintings1.xml*. Do not change the name of the file! An exemplary code looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>
  <sortingImages>

    <welcometxt>
      <txt>Sort Rembrandt's paintings from the earliest to the latest. </txt>
    </welcometxt>

    <image>
      <url>samson.jpg</url>
      <priority>1</priority>
    </image>

    <image>
      <url>ruts.jpg</url>
      <priority>2</priority>
    </image>

    <image>
      <url>tulp.jpg</url>
      <priority>3</priority>
    </image>

    <image>
      <url>saskia.jpg</url>
      <priority>4</priority>
    </image>

  </sortingImages>
```

In order to create a sorting game with your pictures, you need to change the names of the files within `<url>` tag. The order of the images does not matter, but it is important that you give a priority number to each of your pictures. The first picture to be sorted should have a priority of 1.

Within `<welcometxt><txt>` tag you can change text with instructions for your game.

Requires some programming skills

It is also possible to adjust matching game by changing the main code in *matching.mxml* file. Such adjustments include:

- how the score is counted - in the original version, total clicks influence the score (less clicks = higher score); the formula used to count the score:

*Math.floor((8 * pairsTaken) / totalClicks)*; this can be changed in the code;

- changing the size of images; in the original version they are displayed as 120x120px. The semi-transparent image displayed when dragging is called a *drag proxy*. When changing the size of the image, you might also want to change the size of the proxy in *initiateDrag* function.

2.4 Questionnaire

Does not require programming skills

One of the available templates is a questionnaire. It allows users to create a survey or a test with a random number of questions and answers.

In *assets/questionnaire* you can find *questions.xml*. Do not change the name of the file! An exemplary code looks as follows:

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<test>
```

```
  <welcometxt>
```

```
    <txt>Choose correct answer. When ready, click the submit button!</txt>
```

```
  </welcometxt>
```

```
  <question>
```

```
    <q>Which of the names is correct?</q>
```

```
    <answer value="false">The Anatomy Lesson of Dr. Nicolaes Ruts</answer>
```

```
    <answer value="true">The Anatomy Lesson of Dr. Nicolaes Tulp</answer>
```

```
  </question>
```

```
<question>
  <q>Which of the paintings shows a person with head turned in profile?</q>
  <answer value="true">Saskia in a Red Hat</answer>
  <answer value="false">Portrait of Maria Trip</answer>
  <answer value="false">Portrait of Nicolaes Ruts</answer>
</question>
```

```
<question>
  <q>What was Ruts's business?</q>
  <answer value="false">selling jewelry</answer>
  <answer value="true">selling fur</answer>
</question>
```

```
<question>
  <q>Is Samson naked in Rembrandt's painting?</q>
  <answer value="false">yes</answer>
  <answer value="true">no</answer>
</question>
```

```
</test>
```

In order to create your own questionnaire, you need to state your questions within `<q>` tag and all the possible answers within `<answer>` tags. Do not forget to add a `value="true"` or `"false"` for each answer!

Within `<welcometxt><txt>` tag you can change text with instructions for your questionnaire.

3. Adding a Minigame to the Playlist

To add a minigame to the playlist, simply use `<minigame file="matching.swf"/>` within `<media>` tags, adjusting the name of your swf file.

Requires some programming skills

It is possible to add to the playlist more than just one minigame of a kind and it can be done in few steps (the example is for a matching game, but works the same for each minigame):

- 1) Create your first matching game.
- 2) Find the file *matching.swf* and rename it to *matching1.swf*. From now on this is your first matching game.
- 3) Open *matching.mxml* file and find a line with url for a HTTP Service (one of the first lines): `url="assets/matching/matchingImages.xml"` . Change the path to *assets/matching/matchingImages2.xml*.
- 4) Using *make_matching.bat*, create your new *matching.swf* file. This step requires using Flex SDK.
- 5) Store all your images in the usual *assets/matching* directory (all the files for all your matching games should be in the same folder). This is also where the xml file should be stored. Make sure that the xml file for your new matching game is called *matchingImages2.xml*.
- 6) You have now two matching games! The first one saved as *matching1.swf* (using *matchingImages.xml*) and the new one as *matching.swf* (using *matchingImages2.xml*). Now you can easily add them both to the playlist or create a third minigame!