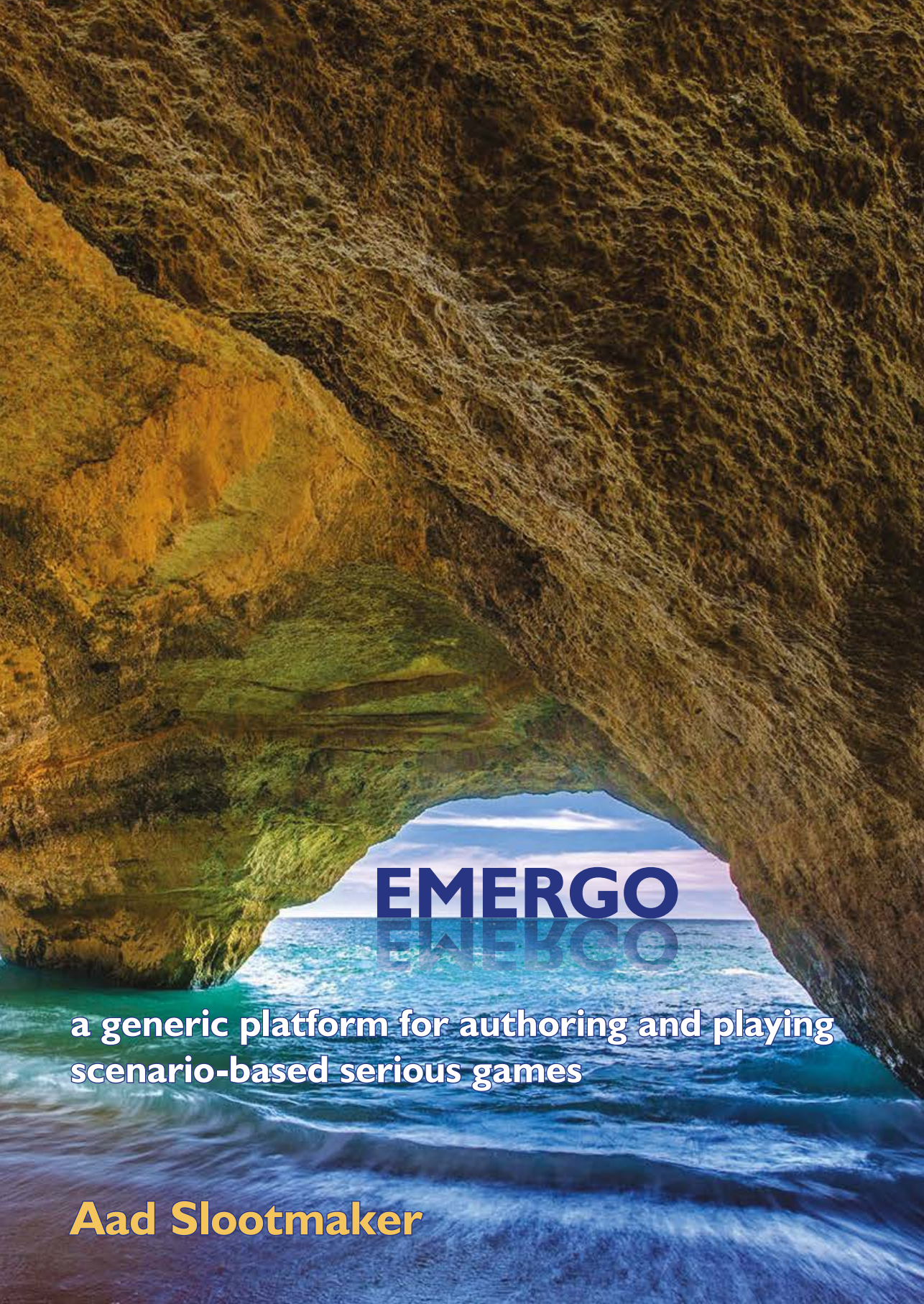

a generic platform for authoring and playing scenario-based serious games

Aad Slootmaker

The research reported in this thesis was carried out at the Open University of the Netherlands at the Faculty of Psychology and Educational Sciences

SIKS Dissertation Series No. 2018-21
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover design: Sneldrukkerij Pasklaar, Sittard
Printing: Datawyse | Universitaire Pers, Maastricht
ISBN 978-94-92739-21-6

EMERGO

a generic platform for authoring and playing scenario-based serious games

Proefschrift

ter verkrijging van de graad van doctor
aan de Open Universiteit
op gezag van de rector magnificus,
prof. mr. A. Oskamp
ten overstaan van een door het
College voor promoties ingestelde commissie
in het openbaar te verdedigen

op vrijdag 14 september 2018 te Heerlen
om 16:00 uur precies

door

Adriaan Slootmaker
geboren op 18 maart 1958 te Rotterdam

Promotor

Prof. dr. E.J.R. Koper, Open Universiteit

Co-promotor

Dr. H.G.K. Hummel, Open Universiteit

Leden beoordelingscommissie

Prof. dr. D. Griffiths, University of Bolton

Prof. dr. M.D. Kickmeier-Rust, Graz University of Technology and University of Teacher Education St. Gallen

Prof. dr. H. Drachsler, Open Universiteit and German Institute for International Educational Research

Prof. dr. J.J.D.M. van Lankveld, Open Universiteit

Contents

Chapter 1

General introduction

1 Introduction

In this thesis we describe the design and evaluation of an online scenario-based serious game platform that enables online universities to efficiently develop their own scenario-based serious games and to deliver them to their students.

Online educators have always been interested and involved in integrating new technologies and methods in their curricula to improve the quality, efficiency and effectiveness of their education. Motivation can be considered to be a key aspect of effective learning (Prensky, 2007). A higher intrinsic motivation often leads to a better performance, increased curiosity and more explorative and self-regulated study behavior (Ryan & Deci, 2000; Martens, Gulikers, & Bastiaens, 2004). Increased motivation may also lead to more satisfied students and to lower dropout rates. For institutes that provide online learning it is even more important to integrate new technologies and methods to literally bridge the distance to their students because they lack facilities like lecture halls, laboratories, field trips or internships.

Nowadays higher education in general calls for more competency-based education and workplace learning. This kind of learning often requires acquiring complex professional and academic skills in authentic professional settings. These skills involve cognitive processes that are associated with higher-order activities like problem solving, reasoning, taking decisions or reflecting in context, which take a lot of time and practice to master. As a result, this kind of experiential education often is difficult to organize because these skills require a lot of time to teach, frequent training in different (protected) situations and time-consuming guidance with the risk of guidance not given on time or to a sufficient extent in actual practice. In addition, higher education faces the problem how to ensure the transfer of acquired skills and knowledge to professional practice. Exercise situations that are equivalent to later professional practice will foster this transfer. Serious games offer a solution for aforementioned problems because they may offer built-in personalized teaching and guidance, offer different (protected) situations to practice and resemble professional practice.

The term “serious game” was first used by Abt (1970) to define a board or card game where players pursue serious goals. The term became popular since 2002 when the Serious Games Initiative promoted the use of video game technology and knowledge to improve game-based simulations in public organizations (Djaouti, Alvarez, Jessel, & Rampnoux, 2011). Since 2002 a serious game is no longer seen as a board or card game but as a special kind of video game. Michael and Chen (2006) defined serious games as games that do not have entertainment, enjoyment, or fun as their primary purpose. A game developer’s primary purpose, for instance, could be learning, but also behavioral change to improve health, public awareness of social issues, or marketing. According to the definition of Michael and Chen (2006) entertainment, enjoyment and fun are subordinate to this primary purpose. However, Dörner, Göbel, Effelsberg, and Wiemeyer

(2016) do not rank the purposes of a serious game by their importance and define a serious game as “a digital game created with the intention to entertain and to achieve at least one additional goal (e.g., learning or health)”. The additional goals of a game developer are sufficient to categorize a game as a serious game and to characterize its type. The authors rightly point out that entertainment games may also be used seriously, e.g., to improve one’s skills.

As our context is online universities we assume the primary purpose of serious games to be learning. We comply with the “serious game” definition of Michael and Chen (2006) where the primary purpose of a serious game is not entertainment, enjoyment or fun. We believe that entertainment and fun should be subordinate to learning and should naturally arise during game development and not be imposed. According to Prensky (2002), the primary objective of a game designer is not entertainment but engagement that is generated by challenge and relaxation. In addition, in games for learning fun is not about amusement but about enjoyment and pleasure, which leads to relaxation and motivation. According to the author: “Relaxation enables learners to take things in more easily; motivation enables them to put forth effort without resentment.”. Franzwa et al. (2014) state that learning and fun should be in balance and that too much fun may distract a player or even stop his progress. We believe that the balance between learning and fun may also depend on the characteristics of the target group, e.g., youngsters or adults, or on the characteristics of a learning task, e.g., more active or more passive.

Serious games can be categorized into different types that correspond to entertainment game types like adventure games, strategy games, role-playing games, simulation games, exercise games, or pervasive games, or may be combinations of several types.
Our focus is scenario-based serious games that combine aspects of adventure, strategy, role-playing and simulation games. This type of games present interactive narratives driven by exploration, problem-solving and decision-making to students playing roles in simulated complex problem spaces that closely resemble professional practice. Students are confronted with ill-defined problems, often allowing multiple solutions and requiring application of necessary methodologies or tools and collaboration with fellow learners (Westera, Nadolski, Hummel, & Wopereis, 2008). We call these games ‘scenario-based’ because the scenario plays such an important role as being the design of the game. The scenario extensively describes the problem space, which activities have to be done and which materials are needed to acquire required complex cognitive skills, and how the problem space should adapt to the individual students. In this thesis we will focus on scenario-based serious games that are aimed at learning.

Serious games are powerful means to provide learning in a more effective way and are able to increase learners’ motivation (De Freitas, 2006; Connolly et al., 2012; Backlund, & Hendrix, 2013; Boyle et al., 2016). In addition, engagement and challenge have a positive effect on learning (Hamari et al., 2016). The primary goal of serious games is to

motivate learners, give them appropriate feedback, improve their skills at the right level, and improve collaboration within groups (Ibrahim et al, 2012). The learner should be in control and learning should be situated and authentic, possibly based on a didactical model or approach and should support transfer of learned skills (Dede, 2009; Clark, & Mayer, 2011; Thillainathan, & Leimeister, 2014). Serious game developers may improve learners' motivation by providing players with an interesting storyline and enjoyable interface, generating interest or curiosity, offering challenging, exciting and customized tasks, giving a feeling of being in control, offering immediate feedback and adapting to the individual learner (Dörner, Göbel, Effelsberg, & Wiemeyer, 2016). These aspects can foster active engagement and can support the intended goals of a serious game.

Given their potential benefits it is no surprise that the use of serious games is still growing and that innovation of the field is stimulated, for instance, by the European Union. In 2015 the global games market grew by more than 9% and the serious games market also showed impressive growth figures (Games Monitor, 2015). In addition, from 2012 to 2015, the number of serious game developers in game studios in the Netherlands grew by more than 7% annually. This growth is still continuing since the video games market is expected to grow at 6.8% annually from 2017 to 2021 (PMC, 2017). Since the first decade innovation of the field of serious games is stimulated by the European Union as is illustrated by numerous European projects such as the ELEKTRA project (2006 – 2008), the PlayMancer project (2007 – 2011), the 80days project (2008 – 2010), the ImREAL project (2010 -2013), the GALA project (2010 – 2014) and the RAGE project (2015 – 2019). The latter project is especially interesting for the field of serious games development because it will deliver self-contained gaming components that can be integrated into existing game engines or platforms to extend their functionality or to foster easier and more efficient game development.

Despite the growth in use of serious games their application within online universities faces a number of issues that are related to differences with the entertainment games sector. First, target groups are smaller in size because a game is generally developed for a specific course in a specific content domain resulting in lower available budgets. Second, in addition to regular game developers, a development team consists of educators responsible for the instructional design and domain experts responsible for the domain content where the latter often differ per development project and thus need to be recruited and reworked each time. Current development tools mostly have insufficient instructional design support for educators and are too complicated for the average educator or domain expert who has no technical background. In addition, serious games have to be playable and testable anytime to enable educators and domain experts to provide correct input during authoring. Third, to enable personalized learning serious games often have to adapt more to the characteristics and needs of the individual player or player group than entertainment games, which requires more game con-

tent to cater for all possible player paths and consequently more complex authoring and more extensive testing. Fourth, player environments have to support this specific type of adaptation by continuous performance monitoring and analysis which requires specific and extensive logging of player data. The same player data should also enable game evaluation for game improvement during development or deployment and enable research on learning effects of a game.
Above mentioned issues that are specifically related to application of serious games call for dedicated development and deployment environments. (i) Smaller budgets ask for efficient development. (ii) Co-authoring by educators and domain experts and adaptation to the individual player ask for specific tooling and more user-friendly user interfaces. (iii) Game adaptation, evaluation and research ask for specific and extensive logging and analysis of player data.

The Open University of the Netherlands (OUNL) is an example of an online university providing bachelor and master degrees in various academic fields. To help students to acquire complex professional and academic skills in authentic professional settings, a scenario-based game platform has been developed and used. As the design of the platform is based on a long experience of developing serious games at the OUNL we first give an overview of this development and the underlying ideas in section 2. In section 3 we introduce EMERGO that consists of a method and a web-based platform and was developed by the OUNL in collaboration with a number of Dutch universities to enable developing and delivering serious games more efficiently. In section 4 we present the design questions that challenged us during the initial and ongoing development of the platform and that we try to answer in this thesis. We end with an outline of the next chapters in section 5.

2 Serious games development at the OUNL

In the eighties the OUNL started developing educational software that was used embedded in its academic courses and was called courseware. In the early years, mainly tutorials and simulations were developed that were based on some kind of physical, mathematical or computational model and were used within natural sciences or mathematics to illustrate the working of these models and to improve their understanding by changing their parameters. This courseware was distributed on floppy disks or diskettes. Personal computers (PCs) were not yet widespread so many students had to visit one of the OUNL's study centers spread around the country, especially if the courseware used video, which required a laser disc player.

In the nineties faculties of law, psychology, cultural sciences and management sciences became interested and involved in development of educational software. The competences to be acquired within their curricula asked for more comprehensive applications in which students should perform complex tasks and didactical and substantive support

and guidance should be integrated. This development resulted in a shift from mainly tutorials and simulations to mainly environments that resembled working environments. A student would be immersed and would work as a trainee, e.g., in a law firm or bungalow park, or as a researcher, e.g., within a public transport company or a psychological clinic. The offered tasks often showed an increase in complexity over time and enabled students to acquire complex cognitive skills. At the time we called this kind of stand-alone applications competence-based multimedia practicals, because different media (text, images, audio and video) were combined to mimic realistic environments and their interaction possibilities (Gerrichhauzen et al., 1998; Hommes et al., 2000; Huysse et al., 1998; Ivens et al., 1998; Leinders et al., 1993; Vandermeeren et al., 1997; Wöretshofer et al., 2000).

Multimedia practicals presented a student's working environments in 2D, used a space metaphor for visiting different locations and experts and usually used video for creating more realism. Occasionally, a mathematical model was used to calculate certain aspects of the environment, e.g., the occupancy rate of bungalows and the influence of marketing campaigns or unexpected events on it. A number of these multimedia practicals were developed with and used by external partners. An internship or a research are large study tasks, which resulted in students spending many hours within these multimedia practicals, sometimes as much as 50 hours. This high study load and the use of different media required an extensive functional design with a scenario describing all student activities and materials needed. In addition, it required a careful planning of development activities and a large investment in time and money. Such an investment could only be justified in case of large student numbers when the investment was earned back during deployment because integrated student support and guidance would result in savings on human guidance.

Multimedia practicals were developed using ToolBook (http://tb.sumtotalsystems.com/) that could be used by ICT developers as well as interaction designers and allowed for efficient component-based development. The components we developed represented different functionalities to be provided to students, e.g., a map or a task overview. Each component had its own authoring and playing part where the authoring part could be realized in a short time by using and connecting standard input elements. Content was usually entered by educators or programmers, sometimes by domain experts. The playing part usually required customization, although reuse of parts of previously developed components was common. Multimedia practicals were extensively tested by students before they were distributed on CD-ROM (Compact Disc Read-Only Memory) or later on DVD (Digital Video Disc) and could be used by students on a Windows PC at home.

These multimedia practicals would now be called serious games, although the engagement and game fun were not determined so much by intense interaction or flashy

graphics but by interesting storylines, challenging tasks and rich content, and the thrill of learning something new.

3 EMERGO

The rise of the World Wide Web in the nineties had no immediate impact on serious games development at the OUNL, because at the time web technologies could not yet provide the rich and direct interaction needed within this kind of applications. The introduction of the AJAX (Asynchronous JavaScript And XML) technology in 2004 and the emergence of frameworks supporting it was the right moment to switch to web-based development and deployment. AJAX allowed for communication between the client and the server in the background, so parts of a web page could be adapted dynamically without having to reload the page after each user action. From now on the World Wide Web would enable us to offer students one equally rich player environment as in the stand-alone serious games before, to offer teachers one authoring environment that could be used for all game authoring, to better support reuse of game content, to simplify delivery of games and updates, also of the platform itself, and to better monitor and support students. Our expectation was that we would be able to develop and deploy serious games more efficiently.

In 2006 four Dutch universities, led by the OUNL, started the EMERGO project (in English EMERGE: Efficient Method for ExpeRiential Game-based Education), an educational innovation project that was co-funded by SURF foundation (the collaborative ICT organization for Dutch higher education and research, https://www.surf.nl/). Based on the OUNL's long experience of developing stand-alone scenario-based serious games, the four participating universities worked together to develop a method and a web-based platform that would enable online universities to develop and deliver scenario-based serious games more efficiently. Main requirements for the platform were to offer a set of reusable and adaptable components that covers most functionalities needed in this kind of games, to provide a user-friendly authoring environment and an intuitive and immersive player environment. In addition, five serious games about environmental decision making were developed that are still in use at two participating universities. During the Skills Labs project from 2008 to 2010, another educational innovation project co-funded by SURF foundation, three institutes, led by the OUNL, developed another four serious games, about water management. In addition, the platform was improved with respect to performance and usability and was extended with new components. During the years thereafter until now the platform has been used to develop and deliver new serious games in cooperation with external partners and faculties within the OUNL. At the moment 26 serious games have been developed. The number of platform components, which represent different platform functionalities needed to acquire

complex cognitive skills, has increased from initially 12 to 30 at the moment. In addition, the platform's performance and look and feel have been improved over the years.

4 This thesis: aim and design questions

The EMERGO project challenged us to design and develop a platform that would enable more efficient development and delivery of scenario-based serious games and would serve all stakeholders involved in development and delivery of games as well as in extension of the platform itself.
We tried to answer the general design question of this thesis:

> *How to design and develop a generic platform for fast and flexible development and delivery of a wide variety of scenario-based serious games that enable complex cognitive skills acquisition?*

The platform should be generic in the sense that it should enable online universities to realize a wide range of game scenarios for different content domains and learning purposes and that it should integrate development and delivery tasks that require different types of platform users in one system. Game development should be fast and flexible. Game authoring should be user-friendly so teachers would be able to author games that way lowering the threshold for developing serious games. Previewing and testing game content and reuse of developed game content should be supported and multiple game authors working in parallel should be possible. Game deployment should also be fast and flexible. Delivery of developed games should be easy, bug fixing of and interfering in already deployed games should be possible and student support in case of technical or functional problems should be supported. To enable complex cognitive skills acquisition the platform should offer a set of common components that covers most of the needed functionalities to acquire this type of skills and should enable reuse of these components and their content in other games. To author these game components teachers would need a user-friendly authoring environment in which they could work independently. And to play developed games students would need an intuitive immersive player environment capable of being adapted to the individual student's progress. In addition, the platform should be sustainable in the sense that it should allow for easy addition of new components and for playing as well as authoring of all previously developed games.

Early evaluations of the authoring environment showed that its usability was suboptimal. Teachers found two platform components, the Script and Conversations components, difficult to use and some of them had trouble to use the Script component independently, despite extensive instruction in advance. This made us question how we could improve the usability of the authoring environment, why components differ in their perceived usability, if our findings would be similar to findings for similar authoring environments and if we could derive some guidelines to improve the usability of this

kind of environments. These considerations led to the first additional design question of this thesis:

1. How to improve the usability of authoring environments for serious games?

Early evaluations of the player environment showed students to be satisfied to very satisfied about its user interface. However, we did not evaluate its usability in detail and were curious whether students were still satisfied, whether we still could improve its usability, whether and how platform components differ in their perceived usability, whether our findings would be similar to findings for similar player environments and whether we could derive some guidelines to improve the usability of this kind of environments. These considerations led to the second additional design question of this thesis:

2. How to improve the usability of player environments for serious games?

5 This thesis: outline of chapters

The general design question is addressed in chapters 2 and 3, the first additional design question in chapter 4 and the second additional design question in chapter 5. We end with a general discussion in chapter 6.

Chapter 2 addresses the general design question with respect to the design and development of the platform and presents evaluation results.

The following main design steps have been taken to answer the general design question.

We identified the intended users of the platform, namely teachers, students, administrators and ICT developers

We set up initial functional requirements for each intended user and non-functional requirements for the platform itself. Teachers should be able to author, preview and test game content, to monitor students and to possibly interfere in running games. Students should be able to play games and to send in outcomes. Administrators should be able to manage platform users, game runs and game teams. ICT developers should be able to rather easily extend the platform with new functionality. The platform itself should be efficient, reliable, stable and usable on multiple operating systems.

Based on the requirements we chose a multilayered client-server architecture, a web-based client, the Java EE platform, Open source frameworks and a centralized database for implementation.

We identified five platform roles that should have their own working environment with associated tasks. The administrator role manages all users and their platform roles. The

developer role authors games, game roles and game components. The run manager role manages game runs and teams and assigns students to runs and teams in a certain game role. The tutor role monitors students and may interfere in a running game. The student role plays games.

We designed a domain model containing all platform entities, e.g., components, games, runs and users, and their mutual dependencies. Our starting point was that the domain model should remain unchanged if new platform components are added, which simplifies extension of the platform. Therefore the domain model is kept simple and does not contain entities that represent game elements such as NPCs (Non-Playing Characters), materials or game rules. Instead, these entities are defined and stored as XML strings in attributes of certain domain model entities. This data-driven approach ensures maximum flexibility.

We identified the initial set of platform components that should be implemented. Our starting point was a typical game scenario that was based on our previous experience in developing serious games. This scenario involved an environment with different locations, interaction with NPCs representing supervisors, colleagues or experts and the use of a tablet with apps that provide background materials, enable communication with NPCs and PCs (Playing Characters) and enable students to acquire needed complex cognitive skills. A Script component should be used to adapt the environment to the individual student.

We devised a generic component design for both initial and future platform components. This design should allow for defining components, their properties, their mutual relations, their content and the structure and properties of their content. All component definitions are stored as XML strings. The same applies to all authoring and student data.

We chose a method to implement authoring of game script. As teachers were expected to do game authoring, programming of game script was out of the question. Popup dialogues are used to assemble game rules consisting of conditions to check property values and actions to change property values. A condition and its associated actions resemble an ‘if-then’ statement in a programming language. A condition is triggered by a change of a property value that is a result of a student action, a timer event or a script action.

After designing the platform we started its implementation. We subsequently implemented the domain model, the various role environments and the initial set of components, all of which demanded their own more detailed design decisions.

Chapter 2 ends with an evaluation of the platform. Evaluated is if the platform indeed caters for more efficient game development, how satisfied teachers and students are

about the platform, and if the platform's functional and non-functional requirements are met.

Chapter 3 also addresses the general design question and is a further elaboration of important aspects of the platform, namely its components, processes, and architecture.

We start with a further elaboration of the generic component design described in chapter 2 and present the general structure of all platform components and their XML definitions that define their allowed properties and content. The components form a flat structure where dependencies on other entities are defined by relations, this way simplifying extension with new components. We also extensively describe all components to illustrate the type of game elements and (didactical) functions the platform supports.

Next, we describe the main platform processes such as the authoring process, the playing process and processes that are supportive or conditional for authoring and playing. We describe the general authoring process for games, game roles and game components and focus on its most important and complex sub-process, the authoring of game component content, which includes game script. A single editor is used for authoring and validation of content. This editor uses a component's XML definition to render the component's content and input elements to manipulate it. Dedicated input elements are used to assemble script conditions and actions. The editor also allows for previewing and testing game content in the player environment in every stage of authoring, while authoring is in progress and from different points within the game scenario. For the playing process we describe the structure of the player environment and its different run components, the rendering of these run components, which involves personalizing content using student's progress, and the event handling process, which includes handling of student, timer, script and peer events. Other platform processes include monitoring or supporting students, managing game runs and populating the platform and game runs with users.

We end chapter 3 with the multilayered architecture we used for the implementation of the different platform components and processes. We describe its various layers with their main components, their responsibilities and the Open source software we used to implement them.

Chapter 4 addresses the first additional design question "*1. How to improve the usability of authoring environments for serious games?*".

To be able to answer the design question we conducted an in-depth qualitative study of the EMERGO authoring environment to determine its usability. Because the number of authors was too small we could not apply quantitative evaluation methods. Instead, we conducted semi-structured interviews (Bryman, 2012) with some experienced game developers who authored a game for higher education and previously authored several

other games. The use of semi-structured interviews allowed us to not only investigate usability but also to get an idea of the user experience, i.e., the usefulness, emotional impact and context of use for a particular user. We prepared the interviews by setting up an interview guide with themes and associated questions. Themes were: the author's general impression of the authoring environment, to what extent the initial requirements for the environment were met, the authoring experience for the used components and the development process. We added the last theme because an insufficient integration of authoring within the development process might influence experienced usability. To answer the design question we compared our usability findings with those found for comparable environments in literature and established usability guidelines to improve authoring environments for serious games.

The use of the authoring environment is embedded in the EMERGO method. The method supports a development team in ideation, the writing of a global game description based on answers to a list of standard questions, the writing of a game scenario in three steps where each step adds more detail and the use of the authoring environment to enter and test the scenario and upload assets. A development team consists of subject matter experts, educational technologists, interaction designers and ICT developers, and may, if necessary, be temporarily reinforced with other expertise.

Multiple usability definitions exist in literature. One of the first and best known is that of Nielsen (1994) who defines usability by its quality of five components: learnability (for novice users), efficiency (amount of time to accomplish task), memorability (for frequent users), errors (number, severity, recoverability), and satisfaction (pleasantness). Nielsen's definition is general and can be applied to all kinds of devices. However, since the EMERGO authoring environment is software and the quality of the software may influence experienced usability, we will use ISO/IEC 25010:2011 (ISO/IEC, 2011) as the theoretical framework by which we will explain our findings. Its product quality model is composed of eight software quality characteristics of which usability is one. Usability is subdivided into six aspects: appropriateness recognizability, learnability, operability, user error protection, user interface aesthetics and accessibility. The seven other software quality characteristics are functional suitability, reliability, performance efficiency, compatibility, security, maintainability and portability. Note that for better readability, we replace 'appropriateness recognizability' with understandability and 'functional suitability' with functionality.

Tool complexity and the type of user may have a negative impact on experienced usability. Entertainment and serious game development usually requires highly complex authoring tools that are used by specialized developers. Hartson and Pyla (2012) and Murray (2004, 2016) identified different complexity types involved. Tool complexity may be related to the interface, i.e., the intricacy or elaborateness of user actions or to the number of editor features and components. However, tool complexity may also be

related to the work domain, i.e., the degree of intricacy and technical nature of the corresponding field of work or to the number of abstract concepts, complex structures or dynamic structures that have to be understood, maintained or tested. Improved usability might surely reduce interface complexity but probably will have hardly any effect on work domain complexity because this type of complexity is inherent to the domain. Experienced usability may also depend on the type of user. For education, Murray (2004, 2016) identified five possible types of users with different complexity capacity. For instance, teachers have a low complexity capacity, so they cannot be expected to use complex authoring tools. On the other hand, ICT developers have a high complexity capacity.

Chapter 5 addresses the second additional design question "*2. How to improve the usability of player environments for serious games?*".

To be able to answer the design question we conducted a mixed method study of the EMERGO player environment. Students from four Dutch Regional Centers for Secondary Vocational Education each played one of two developed games. We used quantitative and qualitative methods to collect both more objective and more subjective and detailed usability data for the player environment. We used a pre-questionnaire just before a game session, individual note taking during the game session, a post-questionnaire just after the game session and a group discussion afterwards. The questionnaires were used to determine students' prior ICT skills (to rule out their possible effect on experienced usability) their SUS (System Usability Scale; Brooke, 1996, 2013; Sauro, 2011) scores and their experienced general usability and component specific usability. The qualitative methods were used to collect as much points of improvement and points of satisfaction as possible regarding the usability of the environment and allowed us to also get an idea of the player experience, i.e., the level of autonomy, immersion, engagement, and challenge. To answer the design question we established guidelines for player environments for serious games.

Usability is a decisive success factor for entertainment games as well as serious games because it is essential for a good player experience. Poor usability may lead to annoyance and distraction and may have an impact on the learning outcomes of serious games (Olsen, Procci, & Bowers, 2011). Although usability is a very important quality factor of a software system, no single definition of usability exists which takes into account all of its possible aspects (Dubey & Rana, 2010). In addition, measuring usability is complex as Lewis (2014, p. 664) emphasizes: "The measurement of usability is complex because usability is not a specific property of a person or thing. You cannot measure usability with a simple 'usability' thermometer (Dumas, 2003; Hertzum, 2009; Hornbæk, 2006). Rather, it is an emergent property dependent on interactions among users, products, tasks, and environments". Validated questionnaires for summative usability evaluation, of which the SUS probably is most widely used (Lewis, 2014), either produce

a general score or scores on general usability aspects, which make them less appropriate to identify more detailed and specific interface related usability issues. Therefore, we also applied qualitative evaluation methods in our study. Just like for the authoring environment we will use ISO/IEC 25010:2011 (ISO/IEC, 2011) as the theoretical framework by which we will explain our qualitative findings for the player environment.
An important concept in the context of gaming is playability. Playability is broader than usability and is defined as "the degree to which a game is fun to play and is usable, with an emphasis on the interaction style and plot-quality of the game; the quality of gameplay" (Usability-First, 2017). Playability may be affected by the quality of the storyline, the degree of responsiveness, the intensity of interaction, pace, control, intricacy, customizability, realism, social and team support, and the quality of graphics and sound.

Usability and playability are both important for games but for player environments for games usability is probably more important than playability. Usability of a game will mainly depend on the usability of the environment's different components. However, playability of a game will mainly depend on playability aspects of the game itself, e.g., the quality of the storyline, feedback, graphics and sound. These playability aspects cannot be influenced by player environments, although the environments should of course support responsiveness and intensity of interaction. Therefore, to answer the design question we will focus on the usability of the player environment. We expect the operation of its components to be rather easy because complex concepts, structures and dynamics (Murray, 2004) that are visible during authoring are hidden for students. However, how components' functions are translated into usable interfaces may leave room for improvement.

Finally, in chapter 6 we provide a review of our results and our main conclusions, recent and future development and research, and significance of the platform.

The development of a platform such as EMERGO requires teamwork. Many have contributed to the realization and extension of the platform. Appendix 1 therefore shows an overview of the estimated contribution of the author to the main development and evaluation tasks presented in chapters 2 to 5.

Chapter 2

Developing scenario-based serious games for complex cognitive skills acquisition design, development and evaluation of the EMERGO platform

This chapter has been published as: Slootmaker, A., Kurvers, H. J., Hummel, H. G. K., & Koper, E. J. R. (2014). Developing scenario-based serious games for complex cognitive skills acquisition: Design, development and evaluation of the EMERGO platform. *Journal of Universal Computer Science, 20*(4), 561-582

Abstract

Serious games are considered to provide powerful and attractive ways to acquire complex cognitive skills for education and training. But existing platforms for development of game-based e-learning often appear either not to be very user-friendly or too rigid or costly. This article addresses the design, development and evaluation of a generic platform for fast and flexible development and delivery of a wide variety of scenario-based serious games that enables complex cognitive skills acquisition. We present the requirements for the EMERGO platform and which common components it offers to cater for most of the needed functionalities within scenario-based serious games. We explain how users in various roles can use the platform to manage, develop, deliver and play a broad variety of scenario-based games. Evaluation data are presented to back up the claim that the platform indeed allows for faster, more user-friendly and less costly development and delivery of scenario-based games. Seven years after the platform has been launched, it until now has proven successful and still continues to evolve. We close off with some conclusions and needs for further development.

1 Introduction

Serious games offer a solution for enabling professional learning at a distance, when the acquisition in actual practice would be impossible or rather hard to realize. Professional education requires students to practice complex cognitive skills in authentic professional settings. These skills involve cognitive processes, e.g., problem solving, reasoning, taking decisions or reflecting in context. This kind of *experiential education* often is difficult to organize in a practical, e.g., because there are more students than internships available or because the supervision of students would be too time-consuming, risky or insufficient in actual practice.

Existing development frameworks for games often are inadequately tuned toward specific learning needs (Nadolski, Hummel, Slootmaker, & Van der Vegt, 2012), and game engines often have been developed for just one specific aspect of a game (e.g., graphical rendering). There are frameworks that integrate a number of these more specific engines, but do not support teachers that well in the process of developing serious games, or have a steep learning curve (De Freitas et al., 2010). For further take-up in education there was a hard felt need to provide teachers with *a user-friendly authoring environment*. Besides this, existing frameworks often lack suitable logging of game progress, which impedes research on the actual effects of serious games.

The Open University of the Netherlands, being a provider of distance education, has a longer experience in developing serious games for complex cognitive skills acquisition in various content domains and with different learning purposes. These serious games were developed on client computers and delivered on CD/DVD. Not all operating systems were supported, delivery was demanding (reproduction), and technical or functional bug fixes could not be delivered easily. And there was little reuse of game components. Games were mostly built from scratch. There was a need for a platform that would simplify and broaden delivery. The platform should further foster reuse and exchange between serious games for different content domains by offering *reusable and adaptable components* for game development.

Developing serious games is often a costly business. Most games are developed as 3D environments requiring a vast investment in 3D graphics that cannot be reused easily in other games. However, use of 3D is not always needed, because maximum fidelity of the environment does not necessarily lead to better learning (Herrington, Reeves, & Oliver, 2007). Furthermore, the development and testing of the didactic scenarios of serious games is quite time consuming, because the intended complex skills require many steps to take and many hours to acquire. There was a need for an approach and platform that would support *more cost-effective* development of scenario-based serious games.

Some 10 years ago the need for a user-friendly authoring environment providing teachers with reusable and adaptable components to develop serious games cost-effective was commonly felt in many higher education institutes. This need then was expressed in the development of a number of online platforms that enabled teachers to develop their own serious games without programming. Examples are Fablusi (http://www.fablusi.com/), Unigame (http://www.unigame.net/) and Cyberdam (http://www.cyberdam.nl/). These platforms enabled the development of multi-role-playing games where learners take on the role profiles of specific characters or representatives of organizations. However, our focus was broader than just role-play. We wanted to offer a rich environment for experiential education where students mostly learn on their own and where other actors are mostly implemented as non-playing characters.

The central *research question* of this article is *how to design and develop a generic platform for fast and flexible development and delivery of a wide variety of scenario-based serious games that enable complex cognitive skills acquisition*. According to Westera (2001) cognitive skills are skills that involve mental processes that occur in the mind while using, transforming or supplementing available knowledge. *Complex cognitive skills* are associated with higher-order activities like problem solving, reasoning, thinking, assessing and concluding. They include the mental processes of analysis, synthesis and evaluation to produce a re-ordering or extension of the existing cognitive structure. *Scenario-based serious games* are games where learners are placed in complex problem spaces, which mimic real world situations. They are confronted with ill-defined problems, often allowing multiple solutions and requiring application of necessary methodologies or tools and collaboration with fellow learners (Westera, Nadolski, Hummel, & Wopereis, 2008). To enable the acquisition of these complex cognitive skills and this type of games the scenario describes the problem space, which activities have to be done, which materials are needed and how the problem space should be adjusted while the student is playing.

To answer the research question, the remainder of this article will be structured as follows. In section 2 we elaborate on the type of scenario-based games the platform supports. In section 3 we present the requirements for the platform. In section 4 we describe how we developed the platform and present the history of versions. In section 5 we present the platform roles, the domain model and common reusable components and their underlying generic design. In section 6 we evaluate if the platform satisfies the requirements and compare it to related work. In section 7 we summarize our findings and present our plans for future work.

2 Scenario-based serious games supported by the platform

Figure 2.1. Screen of a game showing a square with buildings to visit. On the bottom left corner we see an icon for the tablet. On the bottom right corner we see a mike to record parts of interviews and a notepad to make contextualised notes

The platform supports games where the student works as a trainee in an immersive virtual environment that resembles real-life environments like a law firm or an office environment. His virtual supervisor will give him assignments, and will react to and reflect on his outcomes. He will meet virtual experts or other people to gain background knowledge about the skills to acquire. Within the environment the student has a tablet with apps that provide background materials, enable communicating with virtual persons and other students, and help the student to acquire the skills. The student will be confronted with the consequences of his acts. This means that the environment must be able to respond to student actions by giving clear feedback, and adjust itself according to the progress of the student.

Within a game on Sexology, for instance, the student attends two patient interviews and a multidisciplinary meeting, and interviews four subject matter experts. He has to learn to prepare himself for the patient interview, to write a summary of the interview, to work out a model related to the causes of the patient's problem, and to write a proposal for treatment. The student starts the game on a square with buildings related to the Sexology course: a hospital, a university, a school, a health service, an aids center and a station (see Figure 2.1). The station is used to visit virtual patients at home. Within the hospital the student finds his supervisor, subject matter experts, rooms for pa-

tient interviews and meetings, and his own room. He has a notepad to make contextualized notes and a recorder to record parts of interviews. On his tablet the student finds background materials like the patient records, a log containing all notes made with the notepad, an app with all recordings made during interviews, a manual explaining the interface of the game and an email app to get mails and send in assignment outcomes.

3 Requirements for the platform

The *objective* of the platform is to enable the *fast and flexible development and delivery of a wide variety of scenario-based serious games which enable complex cognitive skills acquisition*. Intended users of the platform are teachers, students, administrators and programmers. *Teachers* will develop games by writing a game scenario, selecting relevant educational material and using the platform to enter game data, game materials and game script, and they will monitor students; *Students* will use the platform to play games; *Administrators* will manage platform users; and *Programmers* will extend the platform. Based on our experience and studies carried out by others (Aldrich, 2005), as well as on aforementioned problems with current development, we now list following functional (F) and non-functional (N) requirements for the platform (Table 2.1).

Table 2.1: Functional (F) and non-functional (N) requirements for the platform

Requirement	Description
F1	Offer *teachers* an *intuitive and user-friendly authoring environment* where they independently can *create and edit games*.
F2	Enable *teachers* to *create and edit game roles*, so students playing together in one game can have different roles.
F3	Offer *teachers* a set of *common reusable and adaptable components* that covers most of the needed functionalities to acquire complex cognitive skills using scenario-based serious games. Teachers should be able to *select components* they need and *edit* these now called *game components*.
F4	Enable *several teachers working together on the same game* so work can be divided.
F5	Enable *teachers* to *preview games* or a single *game component* as a student, at any stage of the development process.
F6	Enable *teachers* to *test games* as a student at any stage of the development process and starting from multiple points within the game script.
F7	Enable *teachers* to *import and export games* so games can be distributed to other platform instances and their content can be reused.
F8	Enable *teachers* to *import and export game components* so game content can be reused.
F9	Enable *teachers* to *monitor progress* of students.
F10	Enable *teachers* to *interfere in a running game*, for instance, if outcome quality is insufficient or if a student is stuck in the game.
F11	Offer *students* an *intuitive immersive player environment* where they *play* developed *games*. The player environment should be adjusted according to the actions and progress of a student by using game script.
F12	Enable to *save and persist all student actions*, for game script to operate on, and for evaluation and research purposes.

Requirement	Description
F13	Enable *students* to *send in* assignment *outcomes*, allowing progression within the game (triggered by game script) and monitoring of progress.
F14	Enable *students* to *enrich* the *running game* with user generated content and share this content with other students.
F15	Enable *administrators* to *manage platform users* and their roles.
F16	Enable *administrators* to *manage game runs*, by assigning a cohort of students to a run and assigning students to game roles.
F17	Enable *administrators* to *manage game teams*, teams of students operating within the same game run.
F18	Enable *programmers* to easily *extend* the platform *with* new *languages*.
F19	Enable *programmers* to rather easily *extend* the set of common reusable components *with* new *components*.
F20	Enable *programmers* to *extend* the player environment *with* new *skins,* to be able to offer (external) parties their own look and feel.
N1	Be *reliable and stable*.
N2	Be *usable on multiple operating systems,* e.g., at and across institutes.
N3	Offer *efficient development and delivery of games*. Delivering and updating the platform and developed serious games should be easy and not affect student's progress.
N4	Be *backward compatible*, authoring and playing of earlier developed games should be possible.
N5	Be *integrated with institutional infrastructures*.

Requirements F3 and F11 directly relate to acquiring complex cognitive skills. Learners will perform authentic tasks in an environment that challenges and makes them curious, presents appropriate and unambiguous outcome goals and provides clear, constructive and encouraging feedback (Nadolski, Hummel, Slootmaker, & Van der Vegt, 2012). Requirements F1, F5 and F6 relate to aforementioned need for a more user-friendly authoring environment. Requirements F3, F7 and F8 relate to the need for reusable and adaptable components. Requirements F1, F2 to F8, and N3 relate to the need for more cost-effective development. The requirements are elaborated in a use case diagram (see Figure 2.2).

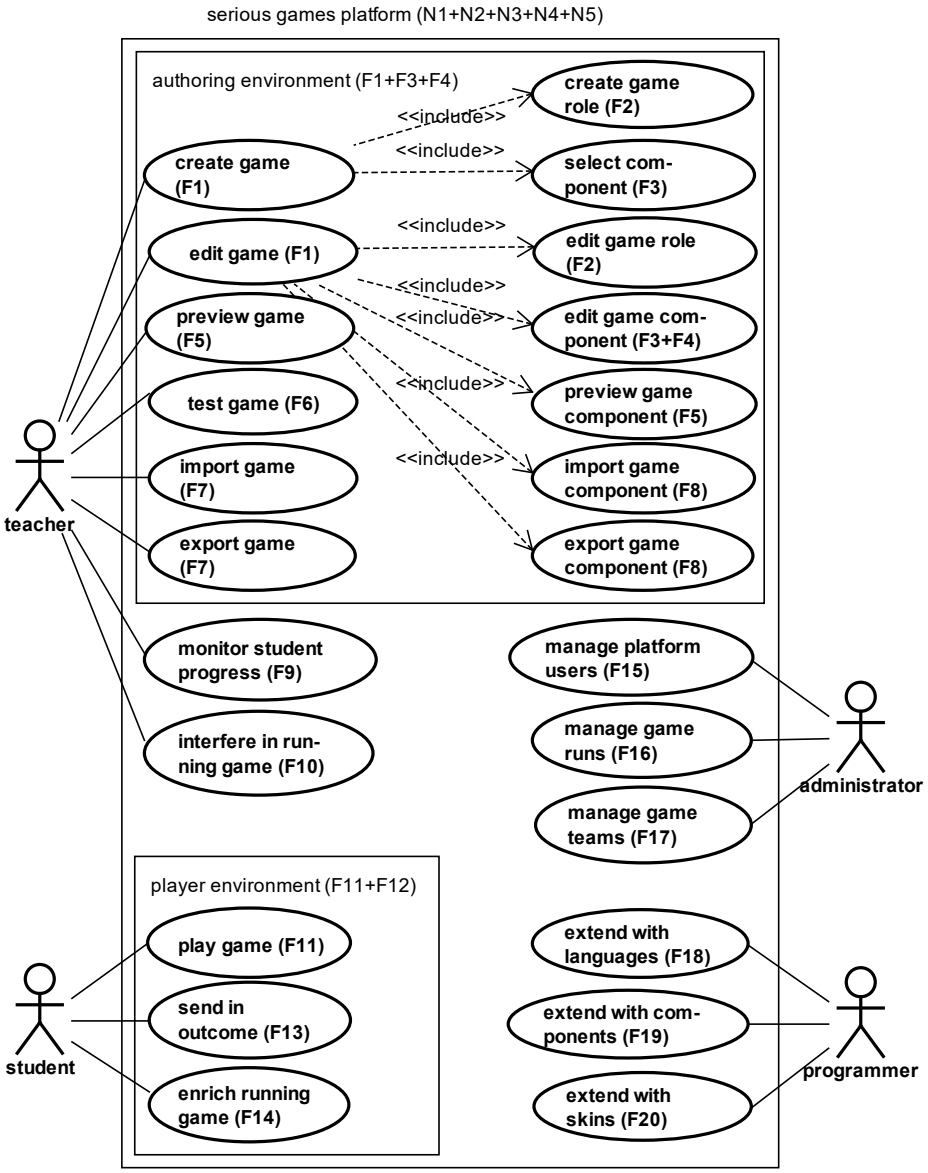

Figure 2.2. Use case diagram for the platform. Requirements are indicated

Rectangles indicate the boundaries of the authoring and player environment. These boundaries are debatable. For instance, previewing and testing a game could be done outside of the authoring environment, but we feel these options should be an integral part of it. In the next section we elaborate on the development of the platform.

4 Development of the platform

Version 1 of the platform was developed within the EMERGO project (2006-2007) that was co-funded by SURF foundation, and was intended to be used by all SURF members. The project had three outcomes: a methodology to support writing the scenario for scenario-based serious games (Nadolski et al., 2008), a platform for developing and delivering the games and five games that were used in education. This article will focus on the *EMERGO platform*. *Version 2* of the platform was one of the outcomes of the Skills Labs project, also co-funded by SURF foundation, and was released in 2010. The project also delivered four games that were used in education. *Version 3* was an outcome of a couple of projects and was released in 2013. The platform is Open source and can be found on SourceForge (EMERGO, 2013).

The first development step was to choose an application architecture. We choose for a multilayered *client-server architecture*, because one layer can be substituted by another implementation without affecting the other layers. To meet requirement F12 (save and persist all student actions) we choose to use a *centralized database* on a server so game script, also located on a server, can operate on student actions, and student data can be shared within multi-role games and is easily available for evaluation and research. To meet requirements N1 (reliable and stable) and N2 (usable on multiple operating systems), and because we had broad experience with it, we choose the *Java EE platform*. To meet requirement N3 (efficient development and delivery of games), we choose the client to be *web-based*, requiring no installation of dedicated client software to develop or play a game and enabling easily updating the platform and developed games. To meet requirement N1 (reliable and stable) we choose the *Spring application framework* (Spring Framework, 2013), to implement our domain model and business logic, and the MySQL database server for data persistence in a centralized database. Both are proven technology and widely used within the Open source community. For the client web interface we choose *ZK framework* (ZK Framework, 2013) that runs on all common browsers. ZK framework is a so called RIA (Rich Internet Application) offering the same interactivity and responsiveness as a desktop application, and therefore offered the best guarantee to meet requirements F1 (intuitive and user-friendly authoring environment) and F11 (intuitive immersive player environment). ZK comes with a very rich set of visual components, which offered the best guarantee to be able to build our own components, meeting requirements F3 (common reusable and adaptable components) and F19 (extend with new components). ZK is very fast and Ajax-based, so all student actions can be saved immediately, meeting requirement F12 (save and persist all student actions).

The platform was *developed by a multidisciplinary team* of educational technologists, interaction designers and programmers. For the development process we used an *agile methodology* similar to Scrum, implying always delivering working software, short iterations, quick response to change and close cooperation within the development team.

We started the development process with the *design of the platform*, which involved following five steps: (1) Identify needed *platform roles*; (2) Create a *domain model* for the platform; (3) Identify needed *common reusable components*, meeting requirement F3 (common reusable and adaptable components); (4) Create a *generic component design*, meeting requirement F19 (rather easily extend with new components); and (5) Design the *component for handling game script*, meeting requirement F11 (using game script, the player environment should be adjusted). In section 5 we will present the results of these five design steps.

Next we started the *implementation of the platform*. After implementing the domain model and business logic we could start *implementing the use cases in a certain order*. Most use cases depend on each other, e.g., before you can create a game, you must first be added as a platform user. While implementing the use cases, we also started *implementing components in a certain order*, determined by their mutual dependency and by the priority within the development team. Version 1 of the platform contained an initial set of common components. This set was extended with new components in version 2 and version 3.

The *evaluation of the platform* involved measuring if requirements F1 (intuitive and user-friendly authoring environment), F11 (intuitive immersive player environment) and N3 (efficient development of games) were met. The evaluations of the other requirements were based on our experiences with the users of the platform, ourselves included. Versions 1 and 2 of the platform were evaluated on the aspects of intuitivity and user-friendliness for teachers using the authoring environment to enter data. Both versions were evaluated on the production ratio for developed games and on student satisfaction with the user-interface of the player environment. Besides this, version 1 was evaluated on student satisfaction, and version 2 on the aspects of quality, studiability and effectiveness of developed games as perceived by students. Intuitivity and user-friendliness as perceived by teachers were operationalized by 'the capacity to use the platform components independent without help' and 'the simplicity encountered when using platform components to enter data', respectively. Intuitivity and user-friendliness were measured using a questionnaire containing questions, like 'Were you able to use the component independently?' and 'How simple was it to use the component?'. Production ratio (as main indicator for efficient development) was measured by comparing development hours (as were recorded in the project administration) with the estimated or measured study time. Student satisfaction was operationalized and questioned as the appreciation of the player environment. Quality and studiability were operationalized in 22 questions, like 'Were the instructions for performing a task clear enough?' and 'Did you get enough background material to perform a task?'. Effectiveness of developed games was determined by students' grades, in one case also by comparing them with grades obtained in classroom education.

5 Design of the platform

In this section we present the design of the platform; the platform roles, the domain model, the implemented common reusable components, the underlying generic component design and the script component.

5.1 The platform roles

Starting from the use case diagram defined in section 3 (Figure 2.2) we identified *five platform roles* that should have their own working environment within the platform: *administrator, developer, run manager, tutor and student*. The administrator and run manager platform role are best filled in by user 'administrator'. The developer and tutor platform role are filled in by the user 'teacher'. The student platform role is filled in by the user 'student', or if a teacher has a role within the game, by the user 'teacher'. The user 'programmer' has no counterpart as platform role, he has his own development environment to extend the platform.

The *administrator* platform role *manages* all *users* and their platform *roles* (requirement F15). Further he can help students who get technically stuck in a game, by *inspecting* a student's *progress* in the player environment, and *adjusting* his *progress* if necessary (requirement F10). If, for instance, certain materials do not become available for a student, due to a bug, the administrator can make them available.

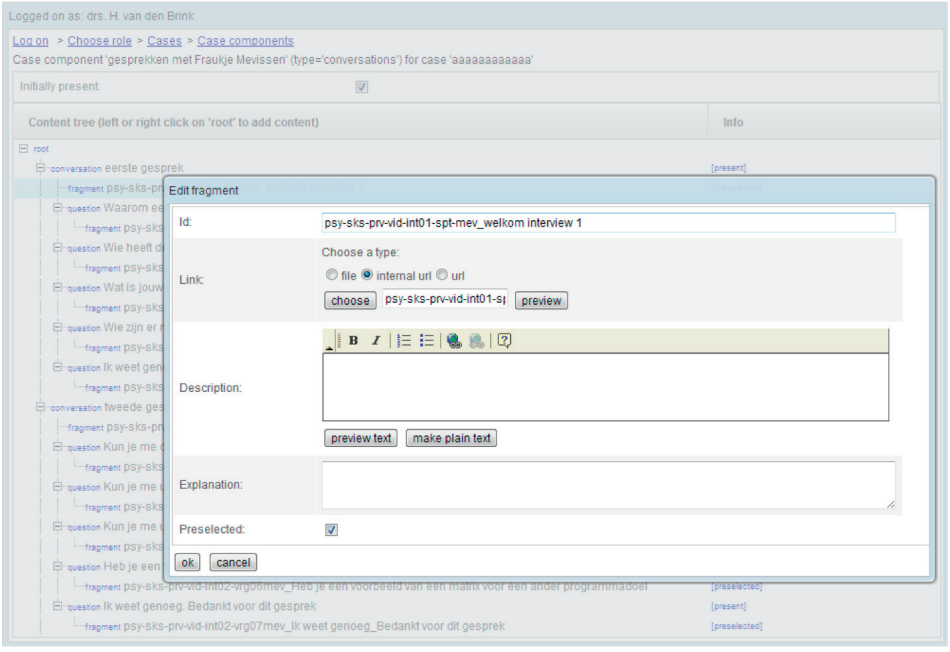

Figure 2.3. Game component content editor showing a dialogue screen to enter a conversation fragment

The *developer* uses the *authoring environment* to create and edit *games* (requirement F1). Per game he can create and edit *game roles* (requirement F2) and *game components* by selecting components to use and enter their content (requirement F3). If needed the *game owner* (the developer who created the game) can *assign* other developers as *author* of certain game components (requirement F4). All game component content is entered using a single editor (see Figure 2.3). During authoring the developer can preview the game or a game component in the player environment (requirement F5). And he can test the game in the player environment from multiple points within the game script (so in time) (requirement F6) and for every game role, and even can test with multiple players. Finally, he can import and export a game or a game component as an IMS content package (IMSCP-IM, 2007) (requirements F7 and F8).

The *run manager* creates and updates *runs* of developed games (requirement F16) and he defines *run users* by assigning users to a run. Further, he can run users to a certain game role and define *run teams* of run users if appropriate (requirement F17).

The *tutor* monitors the progress of students (requirement F9). He gets overviews of tasks students have completed and assignment outcomes they have submitted. If needed, he can interfere in the game by sending an email as if it is sent by a non-playing character (requirement F10), so students do not notice the difference. This way thresholds can be raised, e.g., to guarantee the quality of students outcomes. Further, he can help students who get stuck in a game by *inspecting* a student's *progress* in the player environment and instructing how to proceed (requirement F10).

The *student* sees an overview of games to play and can start the *player environment* (see Figure 2.1) with a chosen game (requirement F11). The player environment renders all developed games in 2D, and mimics the professional practice students later have to work in. All student *progress* is saved and persisted continuously (requirement F12).

5.2 The domain model

The resulting domain model (see Figure 2.4) shows all entities of the platform and how they are related. *Components* are the most important concept of the platform. Components are used to build and play a game. Programmers maintain the set of components and can extend it. *Users* of the EMERGO platform can get multiple platform roles. As an *administrator*, a User can manage Users and give them platform roles. As a *developer*, a User can manage multiple *Games* and is the owner of the Games he creates. Per Game he is the author of multiple *Game Roles* and *Game Components*. He can make other developers author of his Game Components. The Game itself is not much more than a container for Game Roles and Game Components. Components can have multiple Game Component instances and a certain Game Component can be used by multiple Game Roles. As a *run manager*, a User manages *Runs*. A Game can have multiple Runs. The run manager allocates Users to a Run as *Run Users*. He also can create *Run Teams*

of Run Users. As a *tutor*, a User can monitor Runs. As a *student*, a User can participate in multiple Runs as Run User and can be member of multiple Run Teams. A Run User has *Run User Progress* within a Run and a Run Team has *Run Team Progress*. Note that both types of progress can be present in one Run. Progress is related to a Game Component.

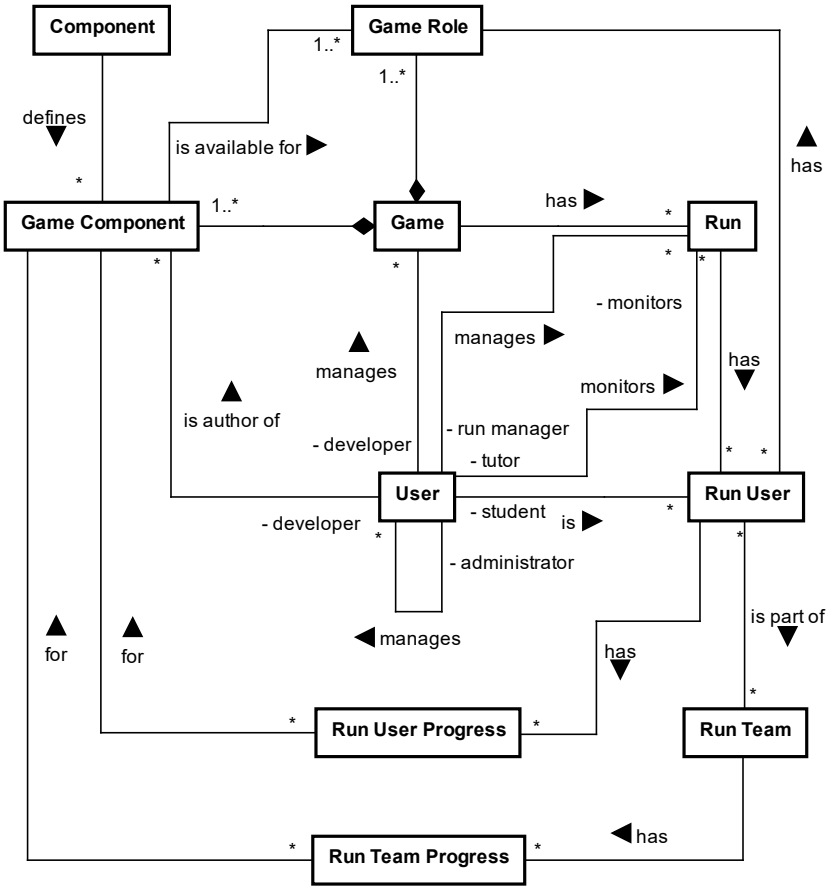

Figure 2.4. Domain model of the platform

5.3 Common platform components

Based on our experience in developing scenario-based serious games over the years, we have identified a number of *components* that represent *common functionalities* for this kind of games. Students are always placed in an environment with multiple locations where they can interview people, and have a virtual tablet with apps to help them with their assignments. Table 2.2 lists all components that we have implemented and in which version of the platform.

Table 2.2: Common components, their function and in which version of the platform they were implemented

Component	Function	Version
Locations	Navigate through the game and stage setting	1
Navigation	More naturally navigate through the game, using hyper regions on location backgrounds and the parallax effect (see Figure 2.1)	3
Conversations	Interact with non-playing characters on location, using video	1
Alerts	Provide popup instructions	1
Notepad	Make contextualized notes. Available on every location	1
Memo recorder	Record parts of interviews. Available on every location	3
Profile	See each other's profile and scores defined in the 'Scores' component. Available on every location	3
Chat	Chat in game. Available on every location	3
Tablet	Provide available apps. Available on every location	1
Assessments	Enable in-game assessment, using items defined in 'Items' component. App on tablet	1
Directing	Examine an interview using different camera angles. App on tablet	3
Email	Enable in-game email, e.g., for providing predefined assignments to students and sending in assignment outcomes by students. App on tablet	1
Google Maps	Enable showing maps with markers. App on tablet	2
Logbook	Provide overview of notes made with the 'Notepad' component. App on tablet	2
Memo player	Look back interview recordings. App on tablet	3
Resources	Provide background material. App on tablet	1
Tasks	Provide tasks overview or to do list. App on tablet	1
Video manual	Explain the player environment interface. App on tablet	3
Items	Provide item bank of multiple choice and multiple answer questions to be used in the 'Assessments' component	1
States	Enable defining game properties that can be read and changed in game script	3
Scores	Enable defining scores to be shown in the 'Profile' component	3
Script	Enable dynamic adjustment of the player environment using game script	1
Relations	Store relations between content of different components	1

The last two components do not represent game functionalities, but are added because they are common in every game. The *Script component* is used by developers to enter the game script. The *Relations component* is used by the platform to store relations between content of different game components, e.g., which items belong to a certain assessment.

5.4 The generic component design

To be able to meet requirement F19 (rather easily extend with new components), we wanted the domain model to remain unchanged if we extend the platform with a component. We therefore choose to store all component related content in XML. It concerns the component itself, the game component content entered by developers and the game component progress of students, as can be seen in the domain model.

To be able to meet requirement F19 (rather easily extend with new components), we had to come up with a *generic design for components*, so components could be added

in the future too. We choose to *define every component by an XML definition* (see example in Figure 2.5), that includes:

1. component *properties* (e.g., a component is present for a student or not);
2. *relations* with other components (e.g., the Logbook component will show all notes entered in the Notepad component);
3. possible *content elements*, that make up the content of a component (e.g., locations, folders, resources, interviews, questions);
4. mutual *hierarchy* of content elements, indicating which content element must be part of another one (e.g., questions are part of an interview);
5. *relations with other content elements* (e.g., an item belongs to an assessment);
6. *content to be entered by a developer* (e.g., the text of a question to be asked or a reference to a video stream to be played);
7. *content to be entered by a student* (e.g., an email text or attachments)
8. content elements' *properties* (e.g., an email is sent);
9. the *type of the properties;*
10. the *default values of the properties*;
11. which *property values can initially be changed by developers*; and
12. which *property values can be read and/or changed by game script.*

```
<data>
  <component type="root">
    <properties/>
    <initial-properties properties=""/>
    <get-properties properties=""/>
    <set-properties properties=""/>
  </component>
  <content type="root" childnodes="alert" max-id="0">
    <alert type="node" id="" key="pid" child-nodes="" multiple="true">
      <pid type="line" private="true"/>
      <richtext type="simplerichtext"/>
      <explanation type="text" private="true"/>
      <properties present="true" selected="false" opened="false" sent="false"/>
      <initial-properties properties=""/>
      <get-properties properties="present,selected,opened,sent"/>
      <set-properties properties="present,sent"/>
    </alert>
  </content>
</data>
```

Figure 2.5. Simple example of an XML definition: the Alerts component

Properties have different *purposes*. There are properties that determine *visibility or accessibility* in the player environment (requirement F11). These properties typically can change during the game and are set by developers, initially or by using game script. Other properties determine the *adaptability* of a component (requirement F3) in either

functionality or layout, and are initially set by developers. Most properties are used to handle *progress* within the game. Adjustments to their values are triggered by student actions (e.g., opening a resource), game script (e.g., sending a predefined email) or the platform itself (keeping game time). We have defined over 30 properties. Table 2.3 lists properties that are used most often.

Table 2.3: Most used properties and their purpose

Property	Type	Purpose	Example
Present	Boolean	Does a student see a component or content element?	A tablet app is present or not
Accessible	Boolean	Can a student access a component or content element?	A door is locked or not
Expandable	Boolean	Can a student expand a content element?	A resource folder can be expanded or not
Expanded	Boolean	Is a content element expanded by a student?	A resource folder is expanded or not
Opened	Boolean	Is a component or content element opened by a student?	A door is opened
Started	Boolean	Is a component or content element started by a student or the platform?	A video stream is started by the platform
Finished	Boolean	Is a component or content element finished by a student or the platform?	An assessment is finished by a student
Sent	Boolean	Is a content element sent by a student or the platform?	An email is sent by a student or the platform

All game component content entered by game developers and all game component progress of students is stored in XML, in a structure defined by the XML definition of the corresponding component. Progress is formed by all property changes in time and possibly associated content like an email text and attachments entered by a student. Some components allow a developer to set properties that enable students to create and share user generated content (requirement F14). This content is saved within progress too.

The generic component design ensures that adding new components has a minimal effect on the authoring environment. Only if a new component demands a new content format, a corresponding input element has to be added in the game component content editor. This, however, does not account for the player environment. It has to be extended with an embedded player for the component.

5.5 The Script component

By using the Script component a developer enters the dynamics of the game scenario, thus determining how the player environment should be adjusted according to the

actions and progress of a student. Conditions and actions are entered using dialogues that require no programming (requirement F1). A condition and its related actions resemble an 'if-then' statement in a programming language (see Figure 2.6).

A script *condition* enables the developer to check whether properties have been set to certain values, e.g., if a student has opened a location then its opened property is set true. A condition can be built up by sub conditions using logical operators. Conditions are triggered by events, either by student actions or timer events, resulting in a property change. If the condition becomes true its related actions will be executed.

A script *action* enables the developer to set a property to a certain value, e.g., a new conversation can be made available by setting its present property to true. When a property is set, the execution of a script action can result in other conditions being triggered. A special kind of script action is the definition of a script *timer*. If its 'parent' condition becomes true, the timer will start. Another condition then can be used to check if the timer fires. Timers have a certain delay, can be defined to be repetitive, and can measure game-time or real-time.

Conditions and actions themselves have properties too. One of them is the present property. By setting its value to true or false a developer can switch conditions and actions on and off, meaning the working of the script itself can be changed too. The Script component only allows conditions and actions to be defined on existing content entered by developers, not on user generated content entered by students.

In the next section we present the evaluation of the platform and its relation to other work.

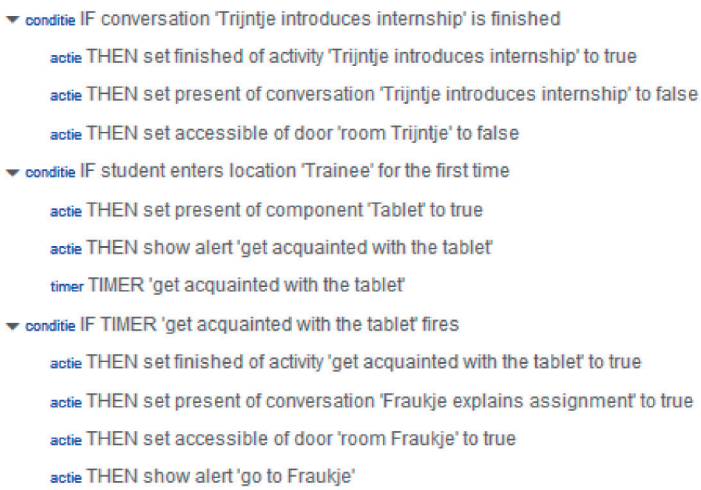

Figure 2.6. An example of script (entered for the game described in section 2). Conditions and actions are added using dialogue popups

6 Evaluation of the platform and related work

6.1 Evaluation of the platform

The EMERGO platform has been used in various projects with both internal and external partners. In seven years, 22 games were developed, which were used in education by nearly 4000 students in total. Games were developed for six content domains, had a broad variety in scenarios and structure and differed both in complexity and study load, ranging from two to 30 hours. Twenty games were single user games and two games were multi-role games that involved collaboration between students. The platform currently is being used by five educational institutes.

We evaluated requirements F1 (intuitive and user-friendly authoring environment), F11 (intuitive immersive player environment) and N3 (efficient development of games) for nine developed games, five running on version 1 of the platform and four on version 2. All nine games were of the same type as described in section 2. The teachers developing with version 1 were different from the ones developing with version 2. Teachers originated from two educational institutes and had a background in Environmental Sciences. Nadolski et al. (2008) evaluated version 1 of the platform and found that teachers only had trouble using the Script component independently (one out of three) and that the Script and Conversations components were most difficult to use. They also found that students (n = 8) were very satisfied with the user interface of the platform and with the developed games. Furthermore, they found an average production ratio of 1:25 (one hour study load costs 25 hours development time) for five developed games, compared with average production rates of 1:100 and higher found before (Alessi, & Trollip, 2001). Version 2 of the platform was evaluated in the Skills Labs project (for evaluation results see http://dspace.ou.nl/handle/1820/2385). Again teachers only had trouble using the Script component independently (one out of four), and found the Script and Conversations components most difficult to use. Students (n = 40) were satisfied with the user interface of the platform. The average production ratio for four developed games was 1:30. Version 2 was also evaluated regarding quality and studiability, and effectiveness of developed games. Students (n = 40) judged quality and studiability of the four developed games as sufficient (three games) or good (one game). Effectiveness was determined by student's grades. The average grade was sufficient to good, only two students out of forty scored insufficient. For one game grades were compared with grades obtained in classroom education, and were slightly better. Evaluation of the other requirements is based on our own experiences with the users of the platform, ourselves included.

Below we discuss if the functional and non-functional requirements were met.

- F1 (intuitive and user-friendly authoring environment) was *partly* met. All teachers could author all components independently, except for the Script component. The Script and Conversations component were quite difficult to use.
- F2 (multiple game roles) was met, but only used in two games.
- F3 (common reusable and adaptable components) was met. One component can be used in multiple games and game components can be imported and exported. The generic component design ensures that components can be defined to be adaptable.
- F4 (several teachers working together on the same game) was met.
- F5 (preview games and game components) was met. It was an indispensable option while developing games and new platform components.
- F6 (test games) was met. It was an indispensable option for fast development of games and new platform components.
- F7 (import and export games) was met. It turned out to be very handy for distribution of games to other platform instances.
- F8 (import and export game components) was met.
- F9 (monitor progress) was met.
- F10 (interfere in a running game) was met. In some games this option was predesigned in the game scenario. However, the option was mostly used by administrators to help students who were stuck in a game.
- F11 (intuitive immersive player environment) was met. Students were satisfied or very satisfied with the player environment.
- F12 (save and persist all student actions) was met. Students almost never lost data and could always continue a game the next session. A first scientific article based on the logging data is in preparation (Westera, Nadolski, & Hummel, 2014).
- F13 (send in outcomes) was met. Outcomes are sent in as an attachment of an in-game email.
- F14 (enrich running game with user generated content) was met. It was implemented for the Resources and Google Maps components.
- F15 (manage platform users) was met.
- F16 (manage game runs) was met.
- F17 (manage game teams) was met.
- F18 (extend with languages) was met. Currently supported languages are English, Dutch and Spanish.
- F19 (rather easily extend with new components) was met. In version 2 and 3, the platform was extended with new common components. The generic component design ensures no or very little adjustment of the authoring environment, although adjustment of the player environment still is time consuming.

- F20 (extend with skins) was met. In version 3, the platform was expanded with the ability to support multiple skins. The current platform has three skins and new skins can be added rather easily.
- N1 (reliable and stable) was met. It is demonstrated by the many games developed and many students playing them.
- N2 (usable on multiple operating systems) was met. The platform currently runs on Windows and Linux servers.
- N3 (efficient development and delivery of games) was met by our choice for a web client, and the abilities to update developed games in case of bugs and to help students who are stuck. Production ratios are better than before.
- N4 (backward compatible) was met. Games developed seven years ago still run on the platform.
- N5 (be integrated with institutional infrastructures) was met. The platform was integrated with the ELO of the Open University to enable single sign-on.

6.2 Related work

During the last decade there were a lot of initiatives to get serious game development on a higher level, strongly supported by the European Commission.

The *ELEKTRA project* (2006 - 2008), for instance, was a research project that focused on bridging the gap between computer science and pedagogy. The project delivered a 3D game on physics meant to engage youngsters for the subject. In-game feedback of these youngsters was used to fine tune the game. The game is analogue to the EMERGO platform in being able to adapt the player environment according to player progress, but differs on being an offline 3D game and not an online development and delivery platform of multiple games.

The *80days project* (2008 – 2010, http://www.eightydays.eu/) was a follow-up of the ELEKTRA project and focused on game adaptation to individual learners, their prior knowledge, abilities, preferences, needs and aims (adaptive personalized learning). Adaptation on a micro level was realized by giving feedback or hinting in specific learning situations, and on a macro level by sequencing and pacing of learning situations tailored to the individual learner. The project delivered a 3D game on geography which was developed using the StoryTec framework (Göbel, Salvatore, Konrad, & Mehm, 2008), an authoring tool for the development of story-based, process-oriented, interactive 3D applications. It resembles EMERGO in enabling authors to develop games without or with minor programming skills. The Story Editor within StoryTec has some resemblance with the Script component of EMERGO in being able to enter conditional transitions within the game, to go from one scene to another, and to enter actions on content elements. And both platforms enable adaptive personalized learning. But while StoryTec focuses on highly graphical oriented 2D/3D games to be developed and played

on a client computer, EMERGO focuses on lesser graphics, use of video and web-based development and delivery. This different focus is related to different customer demands for both platforms.

The *ImREAL project* (2010 – 2013, http://www.imreal-project.eu/), was a European research project focusing on the development of a suite of learning services which extract their data from the real world and can be plugged into virtual environments to augment these environments and enhance self-regulated learning. The learning services were developed by the participating universities. Two existing commercial products were extended to make use of these services. In a first use case an existing role-play simulation environment, developed by EmpowerTheUser (http://www.etu.ie/), was extended to use services related to cultural variations in interpersonal communication, to user generated content, to user profiles (extracted from user activity on the Social Web) and to supporting learners in understanding and improving how they learn. In a second use case another role-play simulation environment, developed by Imaginary (http://www.i-maginary.it/en/), was extended with a story boarding environment for collecting and structuring content for simulations, and the same services as in the first use case. Both commercial simulation environments require no programming, like is the case with EMERGO, and offer rich immersive user experiences, but are not freely available. They support web-based delivery, although it is unclear if all student actions are persisted, but they do not support web-based development. It would certainly be interesting to explore if EMERGO could be extended with the ImREAL learning services.

Another related initiative is the *eAdventure project* (http://e-adventure.e-ucm.es/), a research project of Universidad Complutense de Madrid that delivered the eAdventure authoring tool for the creation of point-and-click adventure games for educational purposes. Developed games can be exported as SCORM package and therefore can be integrated with Learning Management Systems, enabling exchange of adaptation and assessment data. In this respect it is more mature than EMERGO. eAdventure is more focussed on decision making and influencing or adapting certain behaviour, while EMERGO focuses on acquiring complex cognitive skills. Games can be developed on multiple platforms and can be deployed on these platforms and on the web too, although then not all student actions are persisted. It has an easy-to-use game editor, which requires no programming, just like EMERGO, but it does not support multi-role or multi-user games or sharing of content between students.

7 Conclusions and future work

7.1 Conclusions

We demonstrated how to design and develop a generic platform that enables fast and flexible development and delivery of a wide variety of scenario-based serious games which enable complex cognitive skills acquisition.

The platform is *generic* in the sense that it enables a broad variety of game scenarios to be authored, to be played and to be monitored. It offers a set of common reusable components a teacher can pick from to develop a game. The components and their content can be reused in other games. One player environment delivers the variety of scenarios to students and saves and persist all student actions continuously, fostering educational research on all games.

The platform is *fast* in the sense that teachers can use it mostly independent, can draw on already developed components and can preview and test a game during development and from any point in the scenario, which results in more cost-effective development, as indicated by better production ratios than before. Web-based delivery ensures fast and easy delivery of games, updates of games and the platform itself.

The platform is *flexible* in the sense that a game can have multiple authors, a teacher can adjust already deployed games in case of bugs and can interfere in a running game, and the platform provides tooling to help students who are stuck. The platform can be extended rather easily with new components and languages, and skins for the player environment. Developed games can be easily distributed to other platform instances.

Nineteen out of 20 functional requirements were fully met. Requirement F1 (intuitive and user-friendly authoring environment) was partly met. Entering game script turned out to be too difficult. We could improve its interface, but scripting still requires more technical skills so probably better could be entered by a programmer. Another way to improve could be using predefined templates or game patterns, e.g., collaboration scripts (see next subsection). Although requirement F19 (rather easily extend with new components) was met, we expect that extending the player environment can be improved by constructing it using interface building blocks based on macros or templates. All non-functional requirements were met.

7.2 Future work

Collaboration scripts have been scarcely implemented in serious games so far. Therefore, we have built and evaluated two games using online collaboration (Hummel et al., 2010; Hummel et al., 2013). We will use this experience to extend the EMERGO platform with components that support collaboration. This will involve adding new components for rating, voting and negotiation, and extending the Script component to enter

and handle collaboration script. We also consider integrating an online conferencing system as an alternative for chat.

We would like to extend the platform with real-time elements (known as augmented virtuality) like web services for presenting real-time data, real-time video with non-playing characters met in video, and sensor data for better support. With regard to the latter option, at the Open University research is done and software is developed for real time emotion recognition using visual and auditory sensors (Bahreini, Nadolski, Qi, & Westera, 2012). To enable research on the learning benefits of real time emotion recognition in serious games, we will integrate this software with the EMERGO platform, so the player environment can be adjusted according to the student's emotions.

We are involved in some projects where the EMERGO platform will be used in developing countries, e.g., Kenia, Colombia. In these countries connectivity is a problem, so we will make the platform better suitable for low bandwidths. The platform will buffer game content when sufficient bandwidth is available, to account for low connectivity later on. We consider developing a mobile client app for the player environment in case of no connectivity at all. We then could extend the platform to make use of the capabilities of mobile devices like GPS positioning, and making pictures, video and audio.

We already experimented with integrating the Unity Web Player and the EMERGO platform, by playing a Unity game embedded in the platform and exchanging data between player and platform. The platform then could support students playing an existing Unity game. We would like to further explore this promising possibility.

Acknowledgements

We wish to thank SURF foundation for co-funding the development and scaling-up of the EMERGO platform. We also thank all developers, teachers and students of the institutes contributing to the initial development and extension of the platform.

Chapter 3

EMERGO platform components processes and architecture

1 Introduction

In chapter 2 we presented the functional and non-functional requirements for the EMERGO platform and the design steps that led to its realization. The main requirements were to offer a set of reusable and adaptable components that covers most functionalities needed in scenario-based serious games, to provide a user-friendly authoring environment for teachers and an intuitive and immersive player environment for students. We also presented the different platform roles, namely administrator, developer, run manager, tutor and student. The administrator manages all platform components and all users and their platform roles, and may support students by inspecting their game progress within the player environment and adjusting it if necessary. The developer uses the authoring environment to create and author games, game roles, game components and game component content (see Figure 3.1), and can preview or test entered content using the player environment. See section 3.1 for a detailed description of the authoring process.

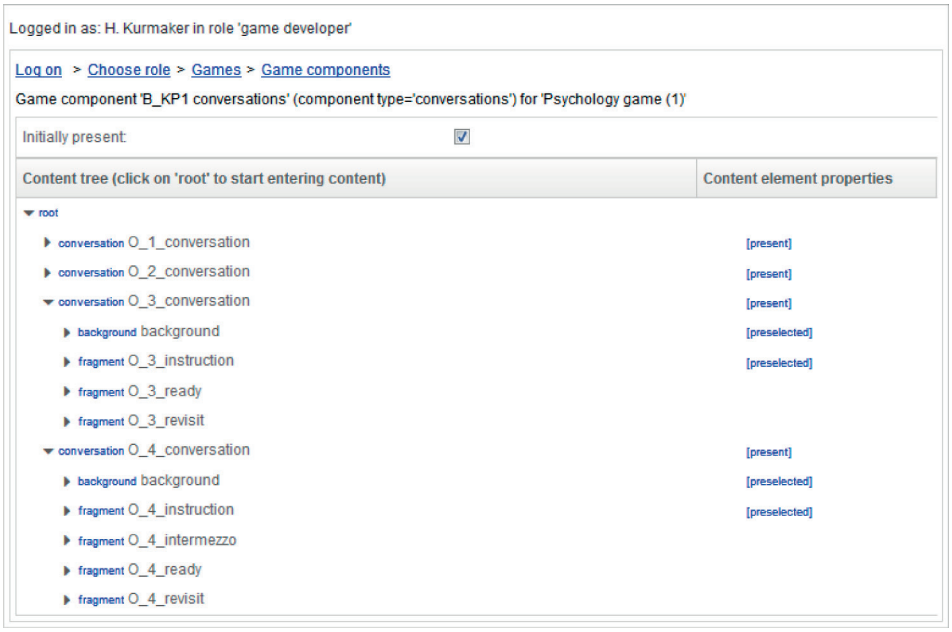

Figure 3.1. The authoring environment's game component content page

The run manager creates or updates runs of developed games, defines run users by assigning student users to a run and a certain game role, and defines run teams of run users if applicable. The tutor monitors the progress of students using overviews and may support them by inspecting their game progress within the player environment. The student gets an overview of games to play and can start the player environment

with a chosen game (see Figures 3.2 and 3.3). See section 3.2 for a detailed description of the playing process.

Figure 3.2. An impression of the player environment for two developed games

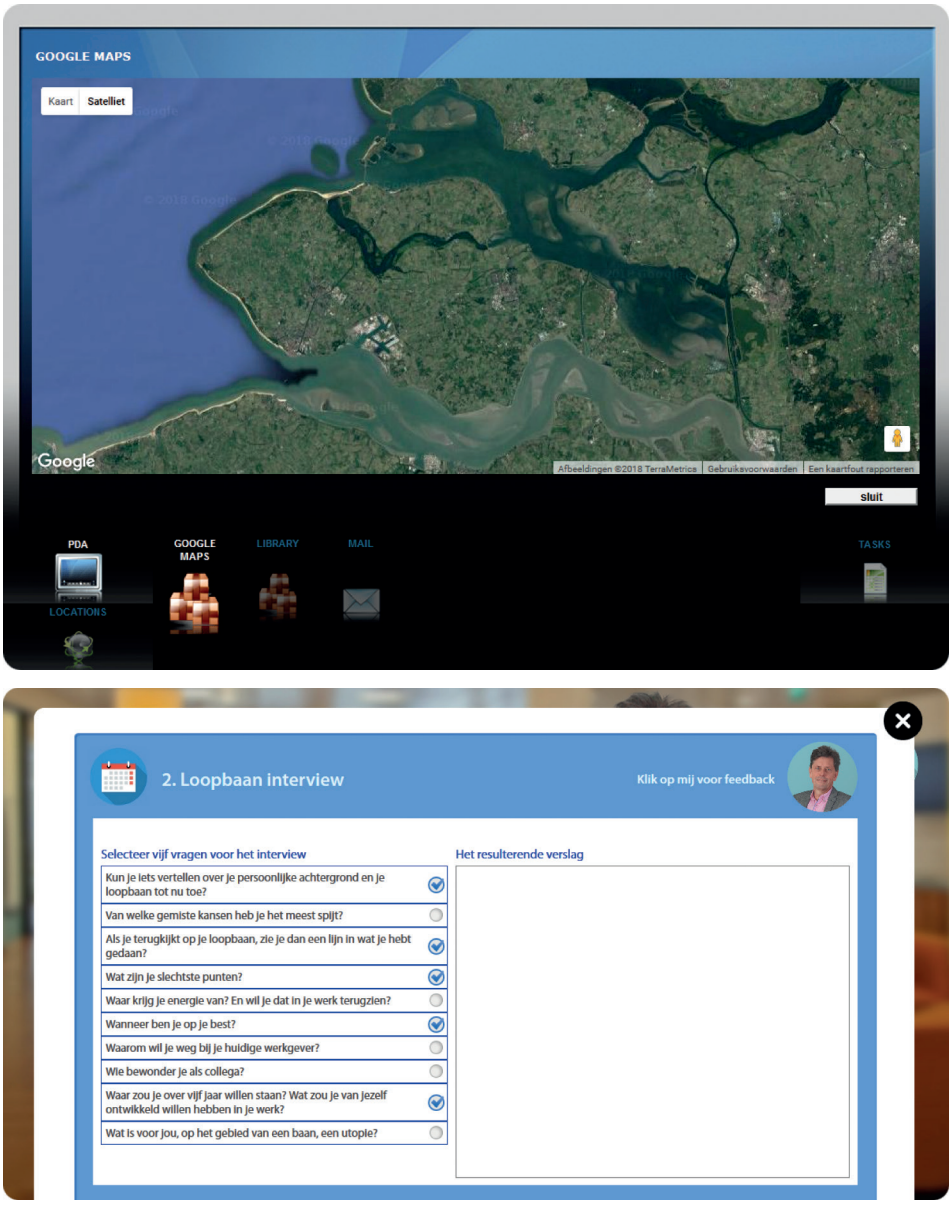

Figure 3.3. An impression of the player environment for two developed games having different skins

In chapter 2 we also presented the domain model of platform entities (see Figure 2.4) with components, users, games, game roles, game components, runs, run users, run teams, run user progress and run team progress. The latter two entities are used to track student or team progress per game component. We briefly presented the differ-

ent platform components (see Table 2.2) and their underlying generic component design (see section 5.4 in chapter 2) which involves component properties, content elements, their hierarchy, their content and their properties, and relations between different game elements. We also briefly explained the working of the Script component, responsible for the dynamics within the game scenario (see section 5.5 in chapter 2). Because the platform should be reliable, stable and deployable on multiple operating systems we chose the Java EE platform (J2EE, 2017) for its implementation. To be able to easily add new platform components the platform's domain model is rather simple and does not contain entities that represent game elements such as materials or game rules. These entities are defined within the platform components' XML (W3C, 2015) definitions that comply with the generic component design and are saved as XML content during authoring or XML progress during playing. The use of XML allows for a very flexible data-driven approach.

In this chapter we will further elaborate on important aspects of the platform, namely its components, processes and architecture.
In section 2 we first present the general structure of platform components and their XML definitions that define their allowed properties and content. Following, we describe all components to illustrate the game elements and functions the platform supports. We start with the Navigation component that is used to define locations and the navigation between them and this way lays the foundation for all other components like the Conversations and Tablet components. In section 3 we present main platform processes such as the authoring process, the playing process and processes that are supportive or conditional for authoring and playing. After describing the general authoring process we focus on the authoring of game component content, which includes game script, because it is the most important and complex sub-process. For the playing process we describe the structure of the player environment after which we focus on the event handling process. This process involves handling of student, timer, script and peer events, which will result in updating the student's progress. The other platform processes involve monitoring and supporting students, managing game runs and populating the platform and game runs with users. In section 4 we present the multilayered architecture of the platform that forms the foundation for the implementation of the different platform components and processes. We describe the different layers, their main components, their responsibilities and the Open source software we used to implement them.

2 The platform components

We start this section with a description of the general structure of platform components and their XML definitions and continue with a description of all components to illustrate all game elements and functions the platform supports.

2.1 The general structure of components and XML definitions

Every platform component represents a certain platform function and has a fixed code, a type, a multiplicity, an XML definition and a possible parent component (see Figure 3.4). The *code* attribute indicates the component's function, e.g., 'navigation' or 'tablet'. The *type* attribute is either 'functional' which means it has a function in the game scenario and can be authored by a game developer or 'system' meaning it is used to support authoring and playing of a game. The *multiple* attribute indicates if a developer may create multiple game component instances per component for a specific game role. For instance, it would be strange to have multiple Navigation, Tablet or Email game components per game role. However, most components allow for having multiple game component instances, which allows for thematically arranging content, e.g., one Conversations component per interviewee or task. The *XML definition* attribute defines the component's possible properties, content and relations with other components or content. Varying availability of components and content is handled by game script (see description of Script component below). The component's *parent component* is either empty meaning the component will be directly available within the game interface or equal to the Tablet meaning the component will be available after opening the Tablet component. The components form a flat structure where mutual dependencies are defined by relations (see description of Relations component below), which simplifies extension with new components.

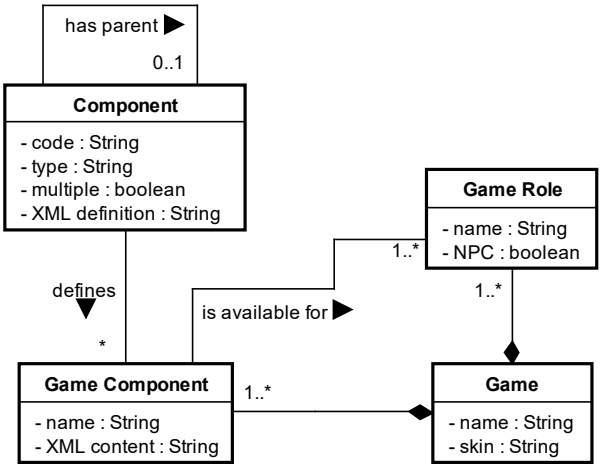

Figure 3.4. Part of the domain model with entities involved in game authoring

The XML definitions define the XML content of game components, which can be seen as instantiations of components within a game (see Figure 3.4). Figure 3.5 shows the general UML class diagram (Fowler, 2004; Object Management Group, 2017) of XML con-

tent, which is a visualization of the generic component design described in section 5.4 of chapter 2. A *game component* has possible *content elements* that may form a hierarchy. Content elements are game elements like, for instance, locations, backgrounds, resources, questions, or fragments. A content element has *content children* that will contain its actual content like titles, names, text or assets. Assets are files like pdf's, images or videos, or URLs. Both component and content elements have *properties* that serve different purposes. See Table 2.3 for a description of most used properties and their purpose. Properties 'present' and 'accessible' determine visibility or accessibility of a game component or a content element in the player environment. The values of these properties may be set by a developer during authoring or changed by game script during a game session. Other properties, e.g., 'correct', 'droppable' or 'position', allow a developer to adapt the functionality, behavior or layout of a game component or content element. Most properties, e.g., 'selected', 'opened', 'started', 'finished' or 'sent', are used to handle progress within the game. Change of property values is triggered by student actions, e.g., when a resource is opened, by game script, e.g., when an email is sent, or by the game itself, e.g., when a video fragment is finished. Both game component and content elements may have *relations* with other game components or content elements, or with properties or game roles (the latter is not shown in Figure 3.5 because it is not defined within the XML definitions.). A relation always involves two objects, e.g., from a specific content element to a specific property. See the component descriptions in section 2.2 for the different kinds of relations. Relations are defined within XML definitions but stored in the XML content of the Relations component, see section 2.2. Script conditions, actions and timers are also content elements, which have their own content children, properties and relations. This allows for game script to switch these content elements on and off during a game session by changing the value of their property 'present' (see Script component in section 2.2).

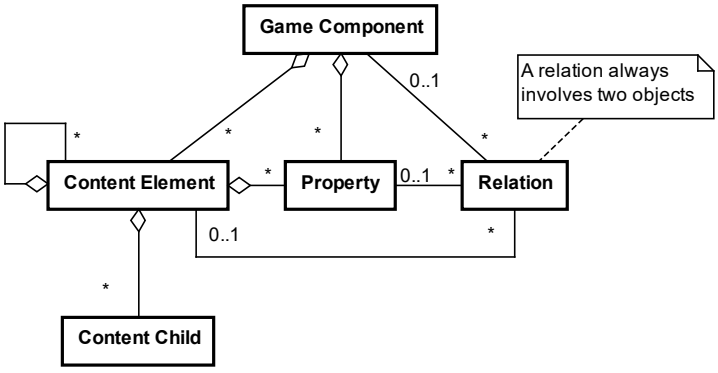

Figure 3.5. General UML class diagram of game components' XML content

Below an example of an XML definition to illustrate its structure, in this case for the Tasks component:

```
<data>
  <component type="root">
    <properties present="true" accessible="true" selected="false" opened="false"/>
    <initial-properties properties="present,accessible"/>
    <get-properties properties="present,accessible,selected,opened"/>
    <set-properties properties="present,accessible,opened"/>
  </component>
  <content type="root" child-nodes="task" preview="true" max-id="0">
    <task type="node" id="" key="name" child-nodes="task">
      <name type="line"/>
      <instruction type="simple-rich-text"/>
      <properties present="true" accessible="true" selected="false" opened="false"
      finished="false"/>
      <initial-properties properties="present,accessible"/>
      <get-properties properties="present,accessible,selected,opened,finished"/>
      <set-properties properties="present,accessible,finished"/>
    </task>
  </content>
</data>
```

In the example above we see that the game component's properties and their default values are defined by the *properties* tag within the *component tag*. If the default property value is 'true' or 'false' its type is Boolean, if it is a number its type is Double and otherwise its type is String. The *initial-properties* tag indicates which properties may be set by a game developer during authoring and the *get-properties* and *set-properties* tags indicate which properties may be inspected or changed by game script during a game session. The game component's possible content elements are defined as children of the content tag, in this case *task*, and are of type 'node'. During authoring the *content tag* is the 'root' element of all content elements. Its *child-nodes* attribute indicates that a task may be added directly under the 'root' element. Its *preview* attribute indicates that the game component may be previewed separately in the player environment during authoring. Its *max-id* attribute is used by the platform to be able to generate a unique id for a newly created content element. The *key* attribute of the task indicates which content child's value functions as a key for the content element, in this case 'name'. This key will be visible for a developer if he creates a relation to a task, for instance, in game script. The task's *child-nodes* attribute indicates that a task may have sub tasks. The task's content children 'name' and 'instruction' are used to enter its content. Just like the component tag, every content element has child tags 'properties', 'initial-properties', 'get-properties' and 'set-properties' to define properties and their default values and if they may be set by a developer or inspected or changed by game script.

Below an example of the definition of a relation, in this case for the Conversations component where a conversation can be present on multiple locations:

```
<data>
  ..
  ..
  <content type="root" child-nodes="conversation" max-id="0">
    <conversation type="node" id="" key="p-id"
    child-nodes="background,fragment,question"
    preview="true">
      <p-id type="line" private="true"/>
      <name type="line"/>
      <ref-locations type="ref" ref-type="conversation-node-to-location-nodes"
      ref-component="navigation" ref-node="location" multiple="true"/>
      ..
      ..
    </conversation>
    ..
    ..
  </content>
</data>
```

In the example above we see that the conversation tag defines a conversation and its *ref-locations* child tag defines relations with 'locations'. Its *ref* attribute indicates that it defines relations and its *ref-type* attribute indicates the relation type, in this case from a conversation element to location elements. It concerns a relation with multiple locations, indicated by the *ref-node* and *multiple* attributes, situated within the 'Navigation' component, indicated by the *ref-component* attribute. Relations are stored in the XML content of the Relations component (see further).

The examples show that XML definitions do not conform to XML schema (W3C, 2016) that may be used to describe the grammar of an XML document. Although the XML definitions together show some similarity to a domain-specific language (Van Deursen, Klint, & Visser, 2000) it was not our intention to develop such a language also because every newly added EMERGO component might require an extension with new language elements. XML schema also does not support validation of the *ref* tag that we use to define relations with other entities. Instead validity of this *ref* tag and all other XML content and students' XML progress is handled by Java components used by the authoring and player environment.

2.2 The description of EMERGO components

We now will describe the structure of the XML content of the different game components in the form of UML class diagrams that show allowed content elements and relations. Content children and properties are left out of these diagrams for clarity. Most content elements are created by a developer during authoring, e.g., locations and backgrounds, but other content elements are created by a student during playing, e.g., notes and chats. We start with the Navigation component that allows for defining locations and the navigation between them and continue with the components that may be present on locations, including the Tablet. Next, we describe all game components that are situated on the Tablet. We end with game components that are not visible to stu-

dents within the player environment and are either used by other game components or for adaptation of the player environment.

Navigation. This component supports student's navigation through the game using hyper regions on locations, which may be decorated. Figure 3.6 shows allowed content elements. Locations define the scenes a student may visit during the game. Every location may have a background that may show a background image. In case of multiple locations or backgrounds the default one is set during authoring, which may be overruled by game script during a game session. Every background may host a number of passages, objects, clickable objects, panels or plugins that all are positioned on the background. A passage functions as a hyper region and is used to go to another location. A developer selects one or more locations, which results in one or more relations with a location. In case of multiple locations a student will select one during a game session. An object is an image that is used for decoration. A clickable object functions as a hyper region and needs a script condition to handle clicking on it during a game session, e.g., to give feedback. A panel is used to play a media fragment. A plugin has to be developed by an ICT developer and allows for filling a specific piece of the background with its own dedicated interface and event handling.

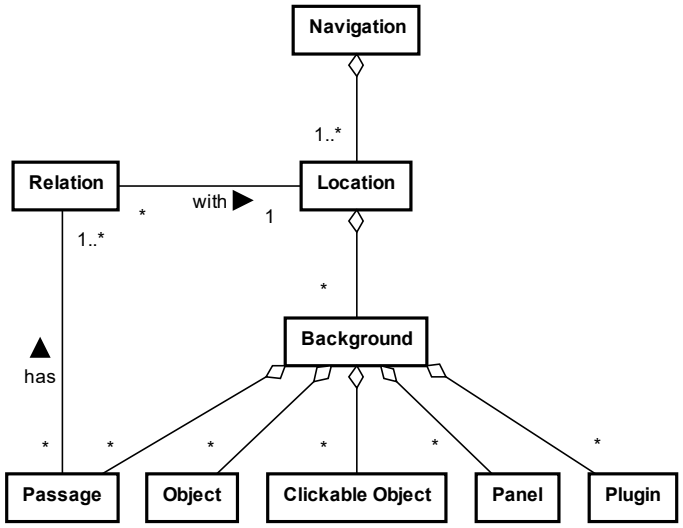

Figure 3.6. UML class diagram of the Navigation component

Conversations. This component enables student's interaction with NPCs (Non Playing Characters) situated at a location but may also be used to play a single video stream, e.g., a presentation, without any interaction. Figure 3.7 shows allowed content elements. A conversation represents a dialogue with an NPC and is opened automatically

when a student enters a location. In case of multiple conversations he will select one. Per conversation a developer selects one or more locations, which results in one or more relations with a location. A conversation may have a background image and may automatically play a fragment, e.g., to let an NPC introduce himself, and may present a number of predefined questions to ask. Fragments support various media formats. When a question is chosen a corresponding fragment is played as an answer. A fragment may be followed by the presentation of a number of other questions, e.g., to zoom in at a subject. In case of multiple conversations on one location or multiple backgrounds or fragments the default one is set during authoring, which may be overruled by game script during a game session.

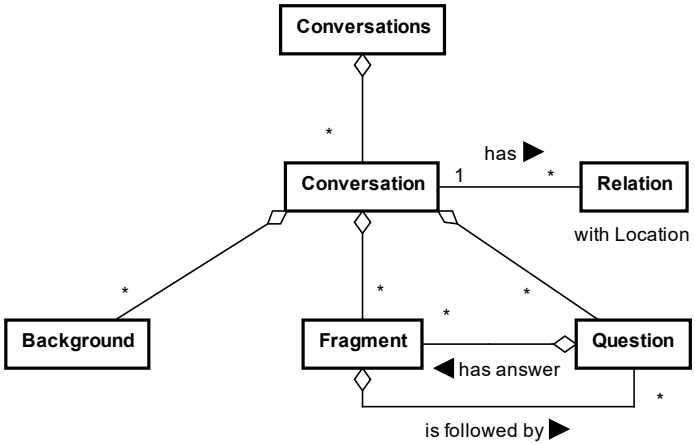

Figure 3.7. UML class diagram of the Conversations component

Notepad. A student can always open this component to make contextualized notes, e.g., about a certain conversation or resource. Allowed content elements are notes (see Figure 3.8).

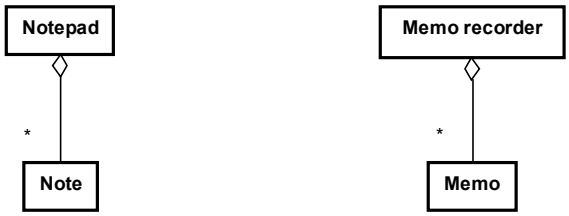

Figure 3.8. UML class diagrams of the Notepad and Memo recorder component

Memo recorder. This component may be available during conversations and allows a student to make recordings of (parts of) conversations. Allowed content elements are memos (see Figure 3.8).

Alerts. This component supports alerting a student during a game session, e.g., to attend a meeting or to insist to work faster. Allowed content elements are alerts (see Figure 3.9) that are authored by a developer. An alert may show a text in a temporary popup or play an audio fragment and needs a script action to show or play it.

Notifications. This component supports providing texts situated on a location, e.g., to show feedback. Whether a new notification replaces an existing one or is added to existing ones is configurable. Allowed content elements are notifications (see Figure 3.9) that are authored by a developer. During authoring a developer selects one or more locations on which the component must be present, which results in one or more relations with a location.

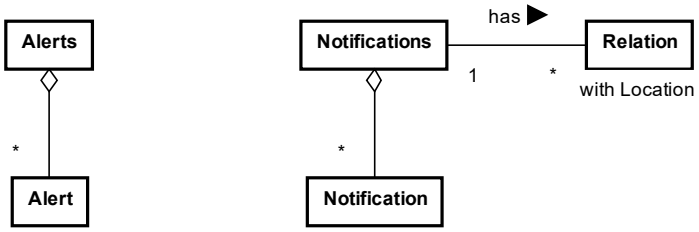

Figure 3.9. UML class diagrams of the Alerts and Notifications component

Scores. This component is available on every location and is used to show student's scores. It permanently shows the most important scores but a click on it shows a score overview of all scores, including the less important ones. Allowed content elements are scores (see Figure 3.10) that are authored by a developer. A score is related to a state (see States component below), which is selected during authoring, and adds a label and a unit to the state value.

Profile. This component can be opened on every location and supports to share profiles and certain scores. Allowed content elements are an image, a mood and scores (see Figure 3.10). The profile image is uploaded and the mood is entered as a text by students during a game session. Just like for the Scores component scores are authored by developers who select corresponding states and decide which scores are shared.

Chats. This component can be opened on every location and supports chatting between students. Allowed content elements are chats (see Figure 3.10) that are created by a student during a game session.

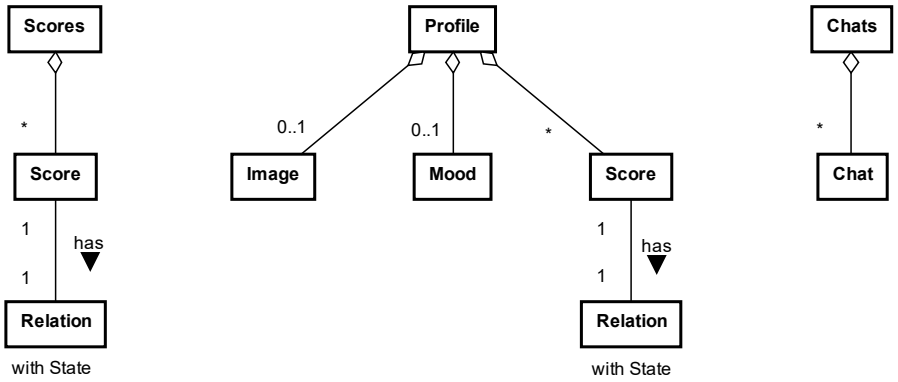

Figure 3.10. UML class diagrams of the Scores, Profile and Chats component

Tablet. This component can be opened on every location and is used to open other components that are situated on it as indicated by their parent attribute (see Figure 3.4). The tablet has no content elements of its own. During a game session style sheets of its app components are used to render icons to open them.

Tasks. This Tablet app is used to show a task overview. Allowed content elements are tasks (see Figure 3.11). Tasks may have sub tasks and are authored by a developer. Whether the completion of tasks is managed by a student himself or by game script is configurable. The availability of other game content needed to perform a task is managed by game script.

Resources. This Tablet app is used to show an overview of resource titles that may be subdivided into folders. Resources can be opened by a student and can be all kinds of assets, like pdf's, images, videos or URLs. Allowed content elements are folders and resources (see Figure 3.11) that are authored by a developer.

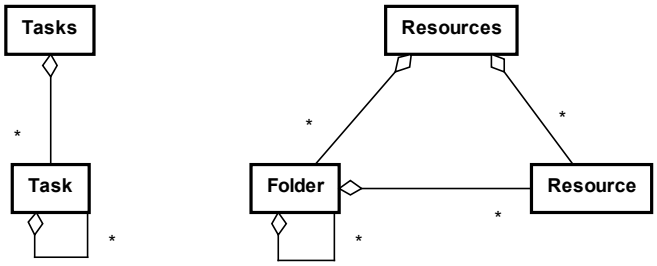

Figure 3.11. UML class diagrams of the Tasks and Resources component

Email. This Tablet app allows students to get predefined mails from NPCs and to send predefined mails to NPCs or PCs. Figure 3.12 shows allowed content elements. A devel-

oper authors mail folders, e.g., inbox and outbox, and can add sub folders to further categorize mails. He also authors incoming mails, which may include attachments, and selects an NPC as sender, which is indicated by the relation between incoming mail and game role. An incoming mail needs a Script action to send it. He also authors outgoing mails, gives them a title and selects NPCs or PCs as receivers, which is indicated by relations in Figure 3.12. During a game session a student will select an outgoing mail, enter its text, add possible attachments and send it.

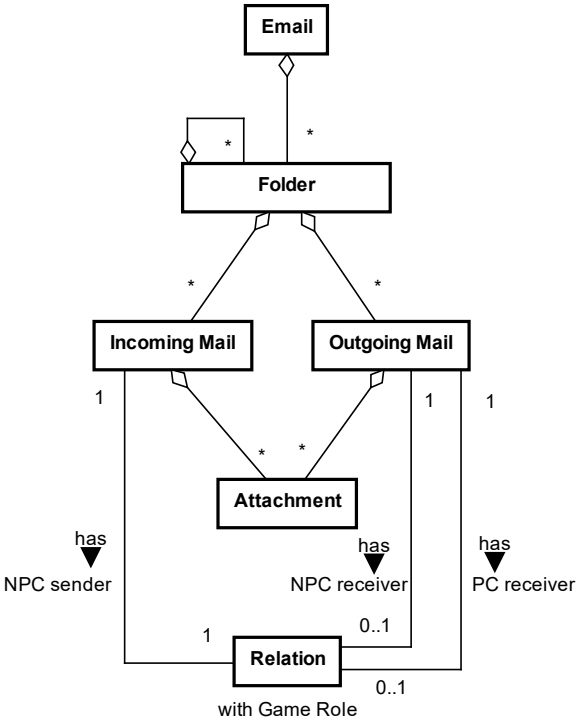

Figure 3.12. UML class diagram of the Email component

Assessments. This Tablet app enables in-game assessment, using items defined in the Items component (see further). Allowed content elements are assessments, instructions, ref items and feedback conditions (see Figure 3.13) that are authored by a developer. If a student opens an assessment a textual instruction may be shown before he starts answering items. 'Ref item' is related to an item (see the Items component) and among others adds a weight to the item relative to other 'ref items' within the assessment. A developer may define a number of feedback conditions to give feedback on the assessment as a whole. A feedback condition consists of a feedback text to show and a condition to check if the assessment is in a certain state, e.g., if all items are answered

correctly or not. A feedback condition may also check properties of other game components or content elements to, e.g., take into account a student's foreknowledge. The condition part of a feedback condition functions the same as a script condition (see the Script component).

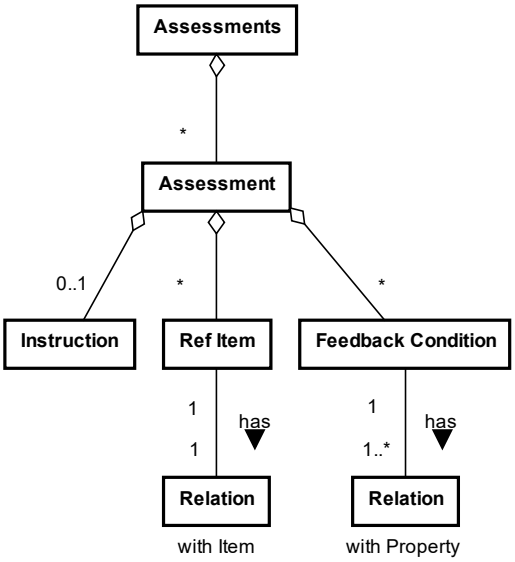

Figure 3.13. UML class diagram of the Assessments component

Logbook. This Tablet app shows an overview of notes, including their context, that are made by a student using the Notepad. In addition, it allows a student to adjust his notes. The Logbook has no allowed content elements (see Figure 3.14) but uses stored content in the Notepad component as indicated by the relation between the two.

Memo player. This Tablet app shows an overview of recordings of conversation fragments, including their context, that are made by a student using the Memo recorder. A click on the recording title starts playing the recording. The Memo player has no allowed content elements (see Figure 3.14) but uses stored content in the Memo recorder component as indicated by the relation between the two.

Figure 3.14. UML class diagrams of the Logbook and Memo player component

Google maps. This Tablet app shows Google Maps decorated with markers that present information about the pointed position when they are clicked. Allowed content elements are markers (see Figure 3.15). The developer determines the initial latitude, longitude and zoom factor of Google Maps for a student and adds markers at certain latitudes and longitudes. Every marker shows a description and may show a resource title. Resources can be opened by a student and can be all kinds of assets, like pdf's, images, videos or URLs.

Directing. This Tablet app allows a student to analyze video recordings of conversations, e.g., a therapeutic session between a practitioner and a patient. Allowed content elements are settings and views (see Figure 3.15). A developer may define various settings that all have a video stream as content. A video stream contains a number of video recordings next to each other that are recorded by various cameras and allows for quickly switching cameras. The views correspond to the various cameras and allow a student to focus on a specific participant or on the whole.

Game manual. This Tablet app uses screen recordings to help students use the various game components. Allowed content elements are fragments (see Figure 3.15) that have a title and a screen recording as content.

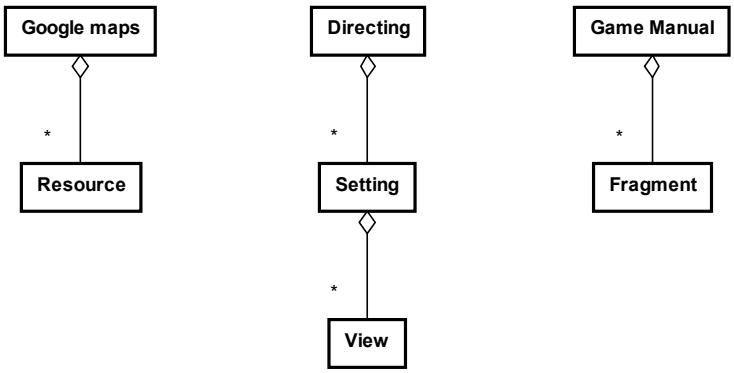

Figure 3.15. UML class diagrams of the Google maps, Directing and Game manual component

Items. This component is used to define question items and feedback on given answers. It is not visible to students within the player environment but is used by the Assessments component that also renders the interface for the items. It functions as a question item bank, which can be used by various assessments. Allowed content elements are items, alternatives, feedback conditions and resources (see Figure 3.16). A developer may define different multiple choice or multiple answer items with a number of alternatives. Feedback conditions consist of a feedback text to show and a condition to check for chosen alternatives. Because the Items component is used as an item bank for

various assessments, feedback conditions within this component are restricted to only check properties within the same game component instance. The condition part of a feedback condition functions the same as a script condition (see the Script component). Items, alternatives and feedback conditions may include resources such as additional learning materials that can be all kinds of assets, like pdf's, images, videos or URLs.

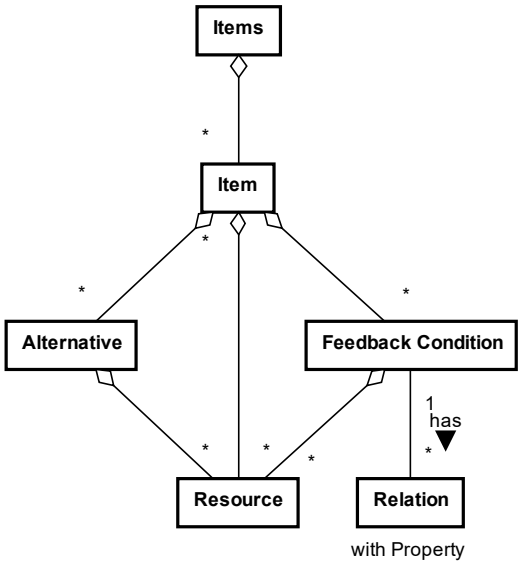

Figure 3.16. UML class diagram of the Items component

States. This component is not visible to students within the player environment but is used to define game states that are used by other components, e.g., to show or store some kind of progress or content. For example, the Scores and Profile components use states to show scores to a student. The Script component may be used to adjust states during a game session. Figure 3.17 shows allowed content elements. A state is used to store some kind of value of type Boolean, Double or String. A formula is used to enter a mathematical formula based on states. The formula is entered as a string where states can be referenced by their key, e.g., '[$score$]' to reference to a state with key 'score'. The string is validated using the BeanShell Java interpreter (http://www.beanshell.org/) to check if it yields a correct Java expression that can be calculated during a game session. A textual content is meant to give access to some textual content of another content element by relating to it, e.g., to a title of a certain task within a Tasks component. Textual content and state values can be embedded in other textual content, e.g., by using '[$task-title$]' in a feedback text to embed a textual content with key 'task-title', which allows for personalized textual content.

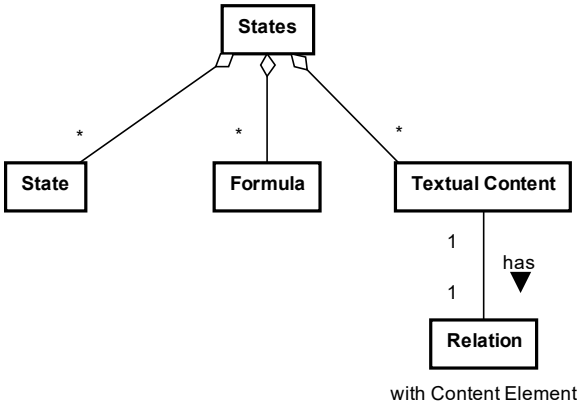

Figure 3.17. UML class diagram of the States component

Script. This component is not visible to students within the player environment but is used to define script to manipulate properties of components and content elements. Figure 3.18 shows allowed content elements.

A developer uses a condition to check whether properties have been set to certain values, e.g., if property 'opened' of a location is set to 'true', and may have sub conditions. The property values to check are stored in a content child of a condition so are no part of the relations between a condition and its checked properties. A condition may be simple and check the value of one property (see Figure 3.25) but may also be compound built up by parts using Boolean logic where each part checks one property. Every part is used to check a property value for certain game roles and for either a game component or a content element. A compound condition may check properties for several game components and content elements. Compound conditions are validated using the BeanShell Java interpreter to check if they yield correct Java expressions. Change of property values due to student actions, timer events or script actions may trigger main conditions, which are situated directly under the 'root' element of all content elements of the Script component. Main conditions may have sub conditions that check other current property values. If a condition evaluates to true its sub conditions will be evaluated and its child actions will be executed. If multiple conditions evaluate to true due to the change of one property value the conditions will be handled in the order in which they were entered.

An action is used to set a property to a certain value for certain game roles and one or more game components or content elements, e.g., to make some conversations available by setting their 'present' property to 'true'. The property value is stored in a content child of an action so is no part of the relations between an action and its set properties. Actions will be executed per condition and in the order in which they were entered. The

execution of an action implies a new change of a property value, which may result in other conditions evaluating to true and their actions being executed, and so on.

Conditions and their actions resemble 'if-then' statements in programming languages because conditions and actions are handled in the order in which they are entered. However, main conditions are not continuously evaluated but only if the value of a property they are related to changes. Conditions, actions and the Script component may be switched on and off during a game session by adjusting their 'present' property, which allows for changing the working of game script itself.

A timer has a certain delay, may be repetitive and can measure in-game time or real time. If its 'parent' condition evaluates to true, the timer will start. Another condition then can be used to check if the timer fires. Actions may be used to stop or restart a timer.

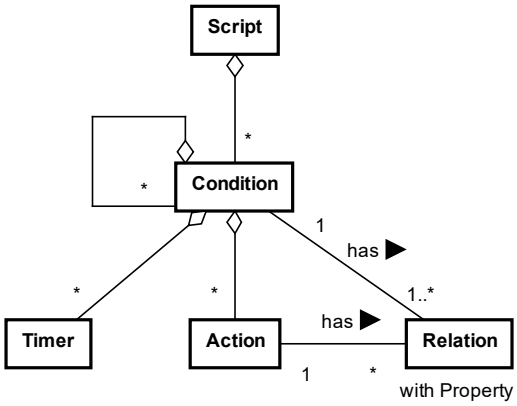

Figure 3.18. UML class diagram of the Script component

Relations. This component is a system component that cannot be authored by a developer. It is used by the platform to store relations defined in other components' UML class diagrams above. Allowed content elements are relations (see Figure 3.19). A relation contains ids of an 'origin' object, e.g., a script condition, and a 'destination' object, e.g., a property. The relation also has a property to store the relation type as it is defined within the corresponding XML definition (see example of an XML definition in section 2.1). In the relations defined so far possible 'origin' objects are game components and content elements and possible 'destination' objects are game components, content elements, properties and game roles. Five combinations of 'origin' and 'destination' objects are defined so far: game component to game component, game component to content element, content element to content element, content element to property, and content element to game role. Examples are, respectively, Logbook component to Notepad component, Notifications component to location element, conver-

sation element to location element, condition element to a property, and incoming mail element to a game role. Another combination that might be possible is from content element to game component, e.g., if the Navigation component would be extended with a content element to open a specific game component. Possible relations with a property or a game role as 'origin' are very unlikely because property is just a basic container without any behavior and game role is no part of game components' content. Instead the relation between game role and game component is defined within the domain model (see Figure 3.4).

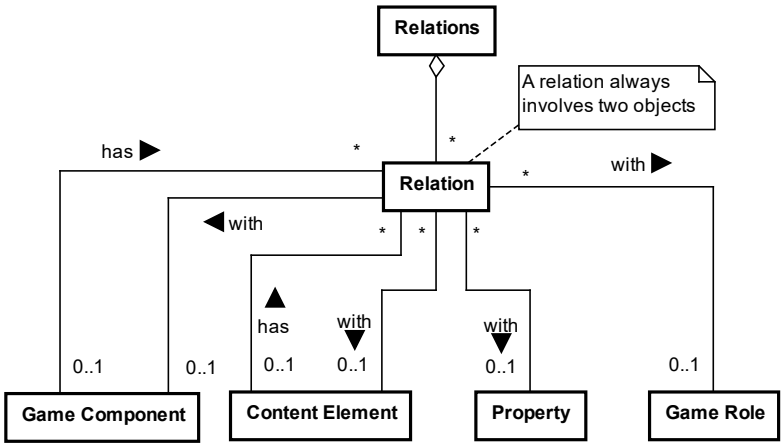

Figure 3.19. UML class diagram of the Relations component

3 The platform processes

The most important platform processes are the authoring process where content of the platform components described in the previous section is entered and the playing process where individual student's progress is kept and saved per component. Therefore, we describe the authoring process, which includes game script authoring, and previewing and testing of game content, and the playing process that includes handling of events, script and cooperation or collaboration. In addition, we describe platform processes that are supportive or conditional for authoring and playing games.

3.1 The authoring process

The EMERGO method involves writing the game scenario (see section 3.1 in chapter 4) and subsequently using the authoring environment to convert the scenario and materials into game content. The scenario among others will describe the PC and NPC game roles, the locations they need, the learning tasks involved and the platform components that are needed.

As the UML class diagrams of the components in the previous section show, there are dependencies between game components that are defined as relations, e.g., the Conversations component depends on locations entered within the Navigation component and the Script component may depend on entered content of multiple game components. However, this does not mean that all content of the independent game components should be entered before that of the dependent game components. As the scenario subsequently describes learning tasks and possible subtasks, content will be entered also subsequently by adding content or script to new or existing game components. So authoring is not a linear process in which game components are created and filled with content in a fixed order. Rather, it is a process of hopping from one game component to another, entering content, previewing entered content and occasionally creating a new game component in between.

Figure 3.20 shows the authoring process. First, a developer creates a game, gives it a name and selects what skin it should use in the playing environment (see Figure 3.4 for game, game role and game component attributes). Second, he creates all game roles, gives them a name and selects them to be PC or NPC. Creating game and game roles are normally one-time sub-processes because game and game roles are defined in the scenario and are not likely to change during authoring. Next the developer will hop between three sub-processes. A developer may (i) create a game component, give it a name and select associated game roles, (ii) enter specific game component content or (iii) preview specific game component content. Because game components have no mutual relation in the domain model (see Figure 3.4) they are created independently of each other. In addition to creating games and game components a developer may also reuse existing games or game components by importing them. Multiple developers may author content in parallel because every game component has its own developer assigned to it. This allows for distributing workload and assigning development tasks to the right team members.

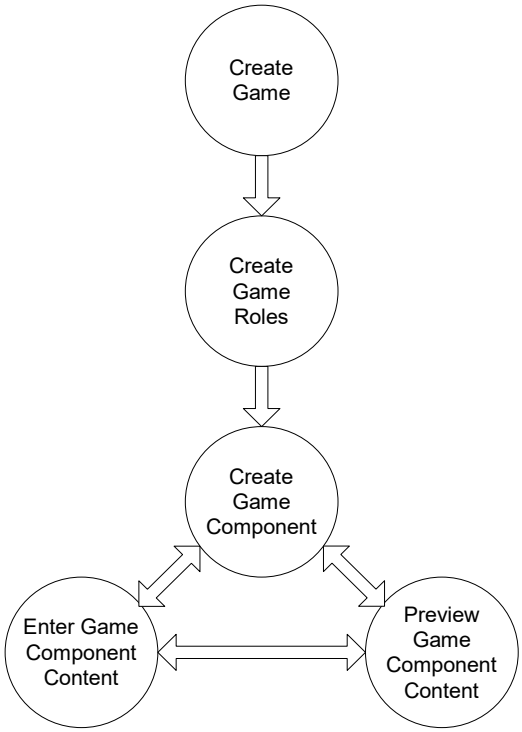

Figure 3.20. The authoring process

Creating a game, game roles and game components are simple less important sub-processes, which take little time. Therefore, we will first focus on the sub-process of entering game component content, which is the most important sub-process and will take most of a developer's time. Entering game component content may be complex because it depends on the complexity of the component and the number of relations with other content. For instance, authoring of the Alerts component is rather simple because it involves only one content element without any hierarchy and with no relations with other content elements. On the other hand, authoring of the Script component is complex because it involves a hierarchy of content elements that have many relations with other content. After describing the sub-process of entering game component content we will describe the indispensable sub-process of previewing this content.

Entering game component content. Figure 3.21 shows the authoring page for game component content that is used for all game components. In this case it shows a page with entered content for the moderately complex Navigation component. The page shows the game component properties to be initially set by a developer (at the top, in this case only one, 'initially present') and a tree structure of already entered content elements that resembles the allowed hierarchy defined within the game component's

XML definition. The tree allows for CRUD (Create, Read, Update and Delete) operations on content elements using a menu and a popup dialogue. It also supports drag, drop, and copying of content elements. A content element is edited in a popup dialogue with input elements that are defined in the XML definition.

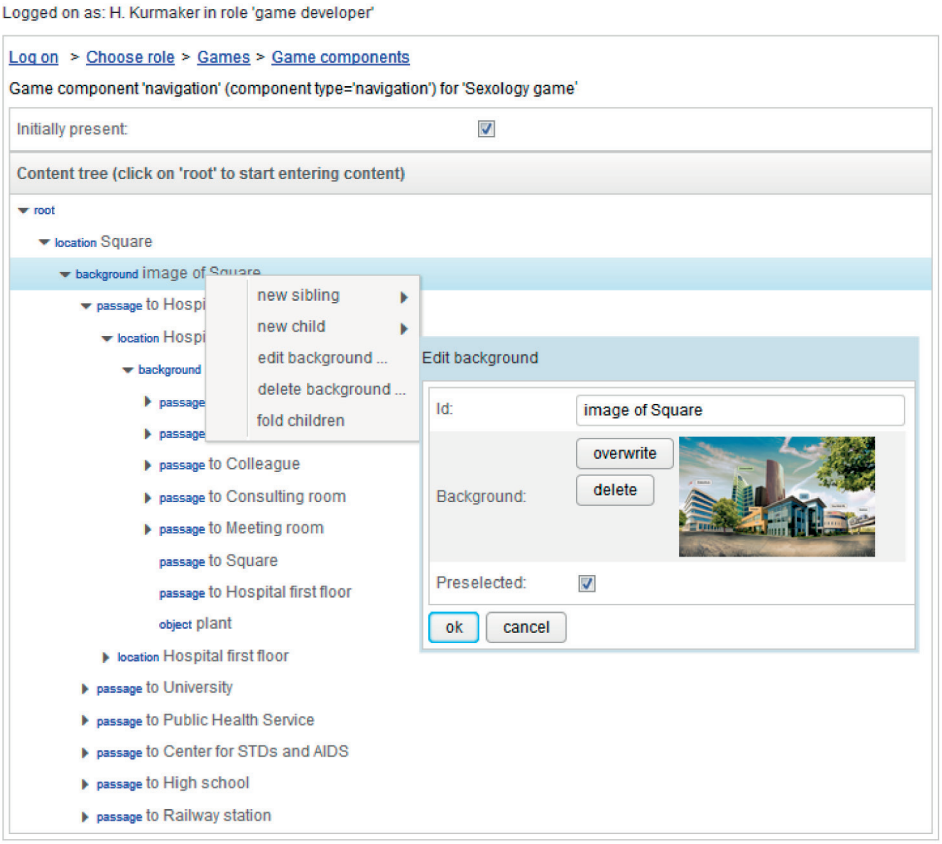

Figure 3.21. Game component content page showing content for the Navigation component

Entering game component content involves a number of sub-processes that we will describe below: (i) rendering of already entered content, (ii) rendering of a popup dialogue to edit a content element and (iii) saving a content element. In addition, we will describe the specific dialogues needed to enter game script conditions and actions.

Rendering game component content page. The UML sequence diagram in Figure 3.22 shows which EMERGO software components are involved in rendering the game component content page. A developer starts by selecting the game component whose content he wants to edit from a list of created game components.

First, the UI (User Interface) layer retrieves the XML content tree for the selected game component from the Content Manager. The Content Manager is specifically used by the game component page and is responsible for the rendering of its content, the input elements in the popup dialogue to author it, the validation of input and the storage of content. The Content Manager uses the Component Manager and the Game Component Manager to retrieve the XML definition and XML content strings for the selected game component. The Component Manager and Game Component Manager are responsible for CRUD operations on components and game components, respectively. Subsequently the Content Manager uses the XML Manager to convert the XML definition and XML content strings into an XML content tree that is a tree structure of instances of a so called XML tag, a Java class that corresponds to an XML tag within an XML string but adds extra functionality. In the process every XML content tag gets a reference to its corresponding XML definition tag. The XML Manager is responsible for managing of all XML entities used within the platform, both during authoring and playing.

Second, the UI layer uses the Content Manager to render input elements for all game component properties that are retrieved from the XML content tree by the XML Manager.

Third, the UI layer uses the Content Manager to render a tree of all content elements where every tree element gets a reference to the corresponding content element. For every content element the Content Manager uses the XML Manager to retrieve specific attributes from the XML content tree that should be visible in the tree such as name and id of the content element.

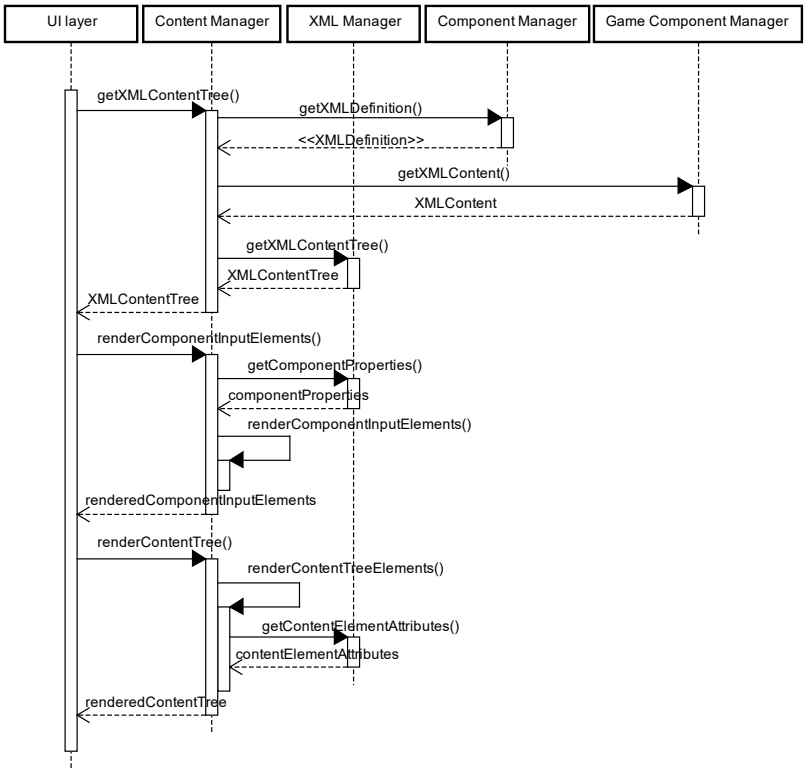

Figure 3.22. UML sequence diagram for rendering the game component content page

Rendering popup dialogue to edit a content element. Figure 3.23 shows the UML sequence diagram for this sub-process. A developer starts by selecting a tree element and selecting the menu option to edit the content element.

The UI layer opens a popup dialogue and uses the Content Manager to render all input elements for the content element referenced by the tree element. The Content Manager retrieves a list of the content element's content and properties from the XML Manager and renders the corresponding input elements. If the content element's content has relations with other game content or assets, the Content Manager uses the Relation Manager or the Asset Manager to be able to set the content of the corresponding input elements. The Relation Manager manages the content of the Relations component (see section 2.2) and the Asset Manager is responsible for CRUD operations on assets.

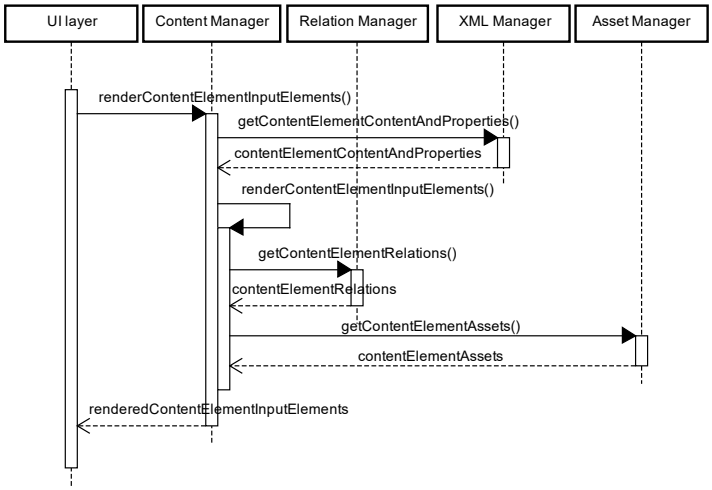

Figure 3.23. UML sequence diagram for rendering the popup dialogue to edit a content element

Saving a content element. Figure 3.24 shows the UML sequence diagram for this sub-process. A developer starts by clicking the 'ok' button to save the content element.

The UI layer uses the Content Manager to validate the content element. Subsequently, the Content Manager uses the XML Manager to validate entered content and properties and return possible validation errors to the UI layer. In case of validation errors the UI layer will render these errors so the developer can adjust his input (not shown in Figure 3.24). Otherwise the UI layer uses the Content Manager to update the content element. First, the Content Manager uses the XML Manager to update the content element with the adjusted content and properties. Second, it uses the Asset Manager to update the content element's assets if applicable. Third, it uses the XML Manager to update the XML content tree (that is kept in memory by the game component content page) with the adjusted content element for the selected game component and to return it as an XML content string. Fourth, it saves this XML content string using the Game Component Manager. Fifth, it uses the Relation Manager to update the relations of the content element with other content. The adjusted content element is returned to the UI layer that closes the popup dialogue and uses the Content Manager to re-render the corresponding tree element.

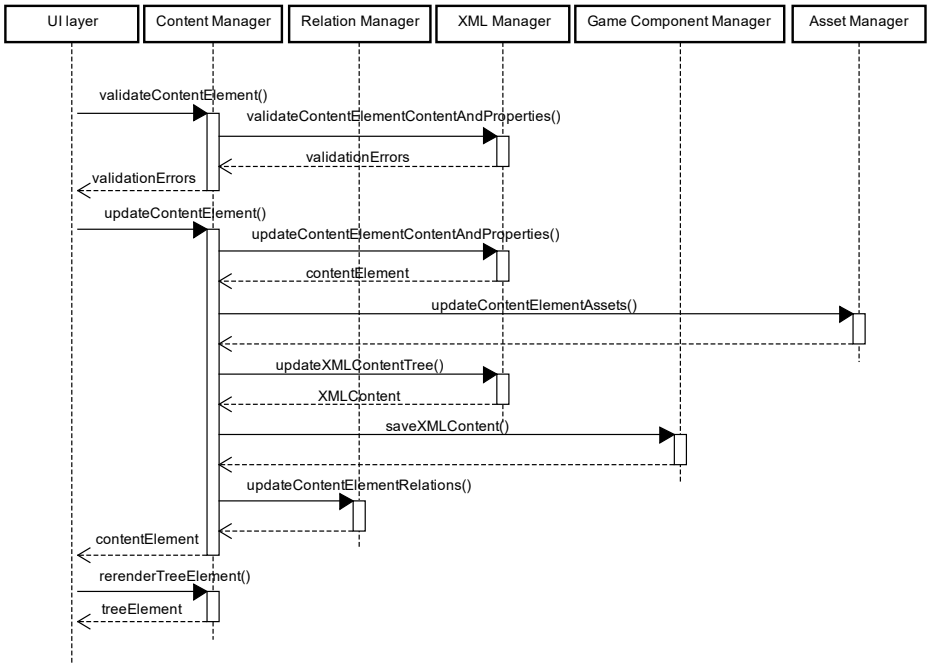

Figure 3.24. UML sequence diagram for saving a content element

Below an example of stored XML content, in this case for a Tasks game component:

```
<data>
  <component>
    <properties present="true" accessible="true"/>
  </component>
  <content id="1" max-id="3">
    <task type="node" id="2">
      <name>Visit doctor</name>
      <instruction/>
      <properties present="true" accessible="true"/>
    </task>
    <task type="node" id="3">
      <name>Visit researcher</name>
      <instruction/>
      <properties present="false" accessible="true"/>
    </task>
  </content>
</data>
```

In the example above we see that component properties 'present' and 'accessible' are 'true' meaning the game component is initially present and accessible for a student. The game component has two tasks where the first one is initially present and the second one not.

Entering game script. Figures 3.21 to 3.24 describe the authoring process of all game component content including game script. The only difference between script and other types of content is that script conditions and actions have specific input elements. Figure 3.25 shows the popup dialogue to enter a script condition that will check if a property is set to a certain value. This popup dialogue will be shown on top of the normal popup dialogue for editing a content element as shown in Figure 3.21. In this case the condition checks if PC 'doctor' has entered location 'Hospital'. Entering a location within the player environment will result in setting the location's property 'opened' to 'true' (see subsection 'Event handling' in section 3.2). After a developer has selected a game component, in this case the Navigation component, he selects for which game roles the condition should be checked and selects the type of content element, in this case 'location'. Out of a list of content elements he selects the ones to be checked, in this case 'Hospital'. He may also enter a pattern to which all content element keys must comply. Next, he selects which property should be checked and selects (for Boolean values) or enters (for String and Double values) the property value to be checked. If the value is a Boolean or a String there is only one operator to check the value, namely the '=' operator. If the value is a Double there are different operators to check the value like '>' or '<'. A developer may also select a function that operates on all property value changes collected during playing. The only function implemented so far is 'count'. It counts the number of times a property is set to a certain value. The developer then may compare this count with a certain value he enters. To enter a script action, which will set a property to a certain value, a similar popup is used. The only difference is that no function or operator can be chosen because these options are irrelevant when a property value is set.

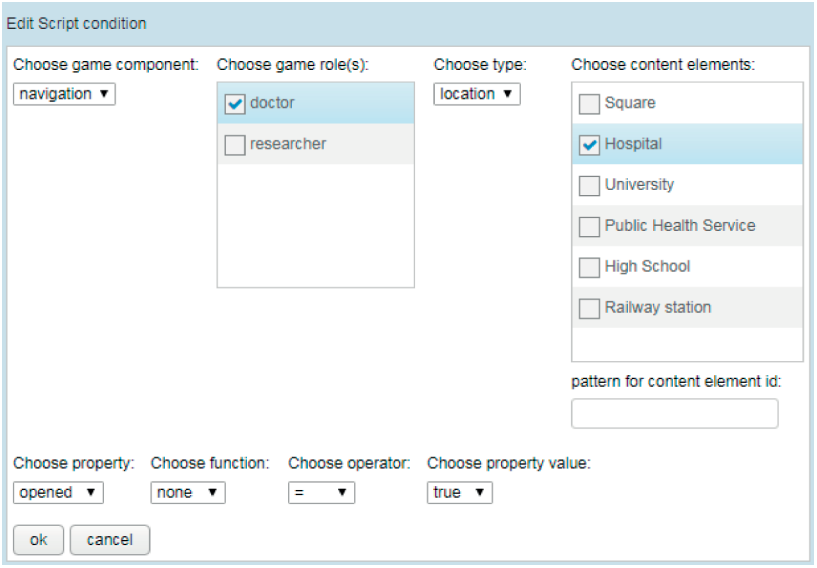

Figure 3.25. Editing a script condition

Previewing game component content. During every stage of authoring a developer may preview his currently entered game component content in the player environment by clicking on a root tree item and selecting the 'preview' menu option, which will open a popup dialogue (see Figure 3.26). Certain content elements like locations and conversations also offer the 'preview' option, which allows for preselecting a location or conversation in the player environment to preview or test a certain part of the game scenario. If a developer clicks the 'preview' button the player environment is opened and his progress will be saved until he closes the player environment. This way it is possible to make 'recordings' up to a certain point in the game scenario. If he subsequently clicks the 'preview read-only' button the player environment is opened with the previously recorded progress. However, this time progress is not permanently saved but only during the play session in the player environment. In this way, the recording may be used to test a specific part of the game scenario over and over again. The recording may also be continued by clicking the 'preview' button again or deleted by clicking the 'delete progress' button. Using the 'new …' button, it is possible to create multiple recordings, which allows for previewing and testing different parts of the game scenario. It is also possible to test cooperation or collaboration between students having different game roles because multiple instances of the player environment can be previewed simultaneously. Recordings can be shared with others by sharing a 'preview read-only' URL of the player environment, which can be used to demonstrate some parts of the game scenario.

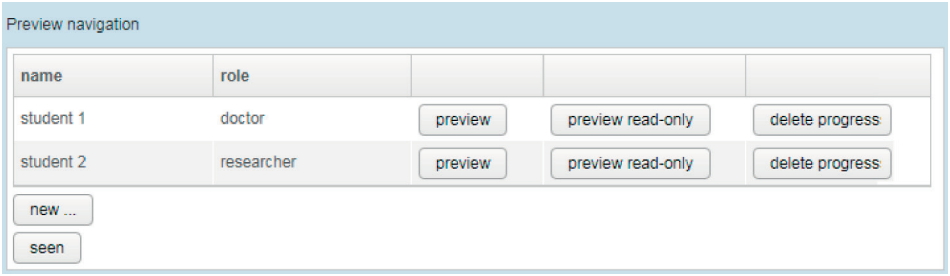

Figure 3.26. The preview popup dialogue to preview entered content within the player environment

3.2 The playing process

The EMERGO platform offers students an intuitive and immersive player environment (see Figures 2.1, 3.2, 3.3, 4.1, 5.1 and 5.2) that is adapted according to actions and progress of a student. Below we will describe the components that make up the environment and the important sub-processes of their rendering, which includes rendering personalized content, and event handling, which includes script handling and progress management.

The UI structure of the environment reflects the flat structure of the platform's components. The UI layer renders a Run Window by using a so called Run Window Java class, which is called this way because deployment of a game involves students being assigned to runs (Tattersall et al., 2005). Every type of game component that should be visible to students is implemented as a child of the Run Window, by using its own specific Java class. All specific classes inherit from a so called Run Component class, which handles their general behavior. From here on, we will use the general term 'Run Component' to indicate a specific implementation. The layering of the Run Components is managed by their style sheets where a higher z-index corresponds to a higher layer (see Figure 3.27). The Run Component for the Navigation component is situated in the lowest layer because its locations form the foundation for the other Run Components. Above the Navigation component we see components that are available on locations. Tablet apps like, for instance, the Tasks and Resources components have their own layer above the Tablet because they are shown on top of the Tablet. The Notepad and Alerts components are situated in the highest layer because making notes should always be possible and alerts should appear above other components. The different player environment skins are implemented as sets of style sheets where style sheet attributes default are inherited from the primary ('OUNL') skin but may be overruled in a new skin.

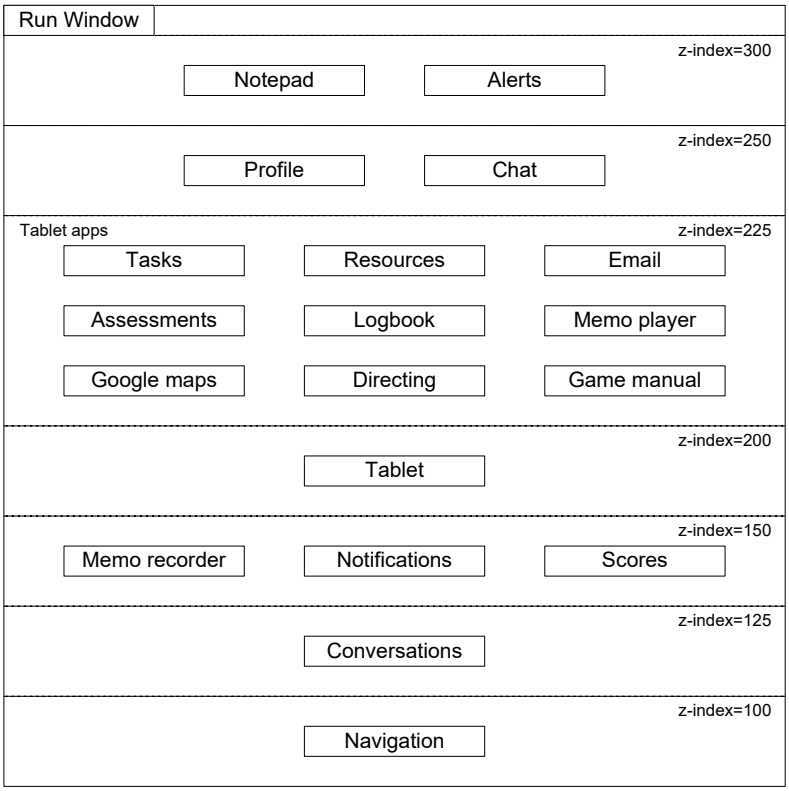

Figure 3.27. Player environment's Run Window with Run Components representing different game components in different presentation layers

The Run Window is responsible for the creation of all Run Components and handles their mutual dependencies. It initially creates an instance of the Navigation component that on its turn will show the default or last visited location. If a conversation is available for this location the Run Window creates an instance of the Conversations component that will show the conversation. The Run Window also will create instances of the Memo recorder, Notifications and Scores components, if applicable. And it will render icons for opening the Tablet, Notepad, Profile and Chats components, if applicable. All Run Components will notify the Run Window if they are opened or closed or if one of their content elements such as a location, conversation or conversation fragment is opened or closed so the Run Window may handle dependencies between different Run Components. For instance, if a conversation fragment or a resource is opened the Run Window will notify the Notepad component to change its notes context. Or if a student navigates to another location the Run Window will close a possibly opened conversation and possibly open a new one. The Run Window also creates a Run Timer that is not visible to students and is used to regularly handle the firing of script timers and peer-to-

peer events in case of cooperation or collaboration (see subsection 'Event handling' below).

Rendering a Run Component. Figure 3.28 shows the rendering process for a Run Component. The process starts if the Run Window creates a new instance of a Run Component, e.g., a Navigation component.

The Run Component will use the Progress Manager to return the personalized content tree for the current student. The Progress Manager is, among others, responsible for retrieving personalized content and storing progress. It uses the Component Manager and the Game Component Manager to retrieve the XML definition and XML content strings for the Run Component and uses the XML Manager to convert these XML definition and XML content strings into an XML content tree of XML tags, just like described for the authoring process. Next, it uses the Run User Progress Manager to retrieve the XML progress string for the Run Component and uses the XML Manager to convert this XML progress string in an XML progress tree of XML tags. The Run User Progress Manager is responsible for CRUD operations on run user progress data. And finally the Progress Manager uses the XML Manager to personalize the XML content tree with XML progress by extending every XML content tag with a reference to its corresponding XML progress tag. The Run Component uses this personalized content tree to render its content and necessary controls and during the process uses the Asset Manager (not shown in Figure 3.28) to get content elements' assets if applicable. Personalized content trees are kept in memory per Run Component for better performance in case updated progress needs to be stored.

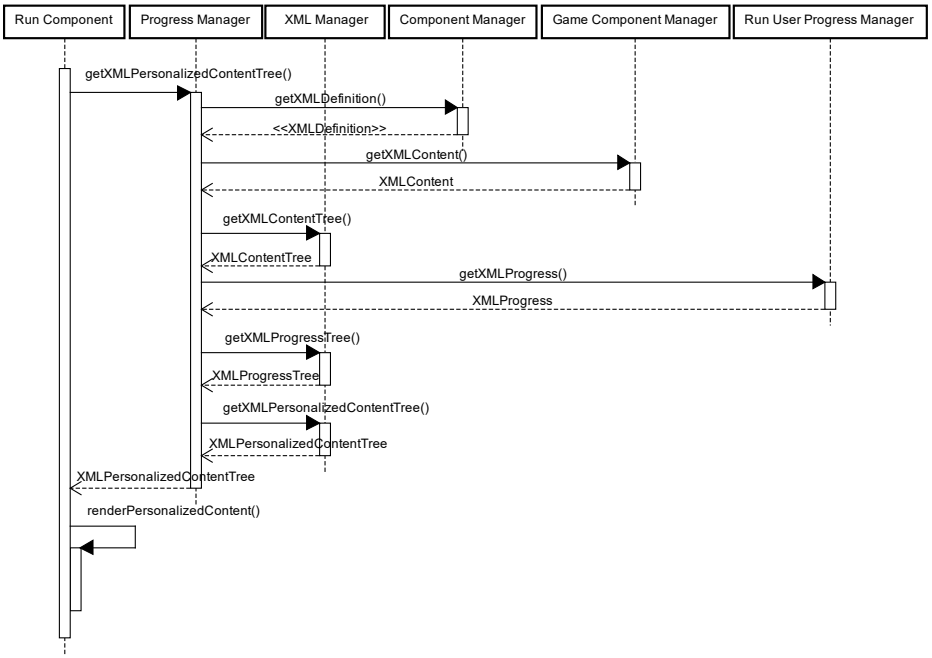

Figure 3.28. UML sequence diagram for rendering of a Run Component

Event handling. Figure 3.29 shows the event handling process for the player environment. It includes the process of handling notify events between the Run Window and specific Run Components, which is described above. Other types of events are user events, property events, timer events and peer-to-peer (p2p) events. Primary input events are user events generated by a student and timer events generated by the Run Timer.

A student generates a user event by, for instance, clicking on a resource title to open a resource. This event is captured by one of the Run Components, in this case a Resources component. It converts the user event into a property event and sends it to the Event Handler (see further for p2p event handling). The Run Component may also send a property event independently, e.g., if a video fragment is finished. A property event has attributes containing the property value and references to the associated property, content element and game component. In case a resource is opened the property value is 'true', the property is 'opened', the content element is a 'resource' element and the game component is a Resources game component. In case of user generated content the property event also contains content children that contain, e.g., an email text, attachments or a chat text.

The Event Handler is responsible for handling of all property events. It forwards the property event to the Progress Manager that will use the personalized content tree for the Run Component in question, which is kept in memory, the XML Manager and the Run User Progress Manager (both not shown in Figure 3.29) to store the changed property value and possible new or altered content children. The Event Handler also forwards the property event to the Script Handler that will evaluate script conditions related to the property event's attributes. If a condition evaluates to true its associated script actions are executed, which results in sending newly generated property events back to the Event Handler. Again, the Event Handler forwards these property events to the Progress Manager to store changed property values and forwards the property events to the Script Handler to evaluate conditions and so on, which may result in a cascade of property events. The Event Handler also forwards every property event to the Run Window that will send a notify event to the Run Component corresponding to the game component attribute within the property event. This notify event allows the Run Component to update itself, e.g., to change the presence of a specific content element.

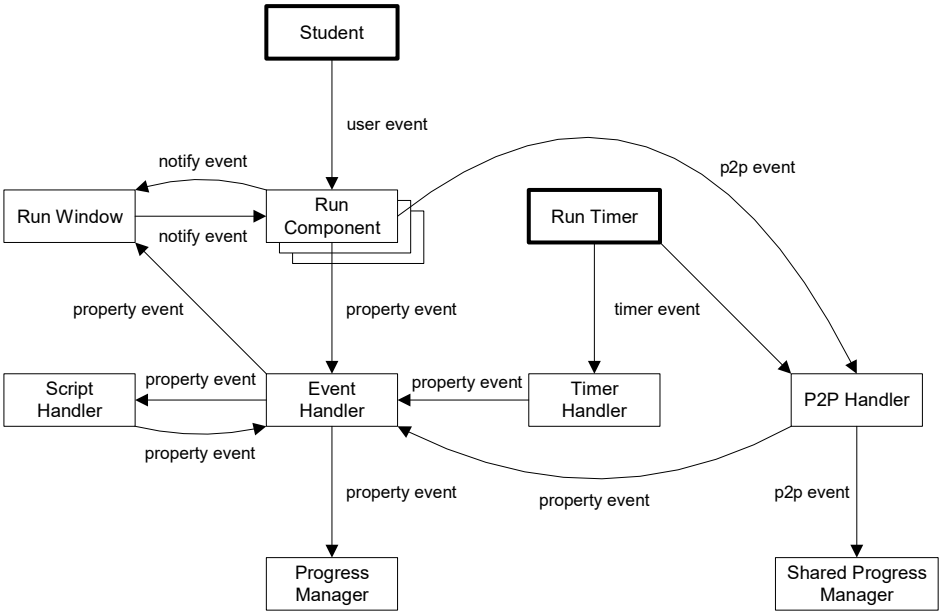

Figure 3.29. Event handling process for the player environment

The Run Timer regularly sends timer events to the Timer Handler that handles script timers and to the P2P handler that handles sharing change of property values to other students (see below). If a script timer fires the Timer Handler sends a property event to the Event Handler. In this case the property event's attributes contain the property value 'true', the 'finished' property a 'timer' element and a Script game component. The Event Handler further handles the property event as described above.

If a change of a property value has to be shared with other students, e.g., when sending an email or chat, the responsible Run Component will send a p2p event to the P2P Handler. A p2p event is an extension of a property event that also contains references to the sender and receiver of the event. The P2P handler sends the p2p events to the Shared Progress Manager that will use the XML Manager and the Run User Shared Progress Manager (both not shown in Figure 3.29) to actually store the changed property value for other students. Timer events sent by a Run Timer in another student's player environment trigger the P2P Handler of the other student to regularly check for p2p events and retrieve them using the Shared Progress Manager. The P2P handler converts possible p2p events to property events and sends them to the Event Handler that further handles the property events as described above.

Below an example of stored student's XML progress, in this case for the Tasks game component:

```
<data>
  <component>
    <properties selected="true,12.118" opened="true,12.122,false,18.385"/>
  </component>
  <content id="1" max-id="3">
    <task type="node" id="2" ref-content-id="2">
      <properties finished="true,46.999"/>
    </task>
    <task type="node" id="3" ref-content-id="3">
      <properties present="true,47.007"/>
    </task>
  </content>
</data>
```

In the example above we see that component property 'selected' is set to 'true' 12.118 seconds after the start of the game meaning the student clicked the Tasks icon on the Tablet to open the Tasks component at that moment. This resulted in the Run Window creating and opening the Tasks component as indicated by property 'opened' set to 'true' after 12.122 seconds. Property 'opened' becomes 'false' after 18.385 seconds meaning the student closed the Tasks component. Note that all changes of property values are saved. Due to game script, property 'finished' of the first task is set to 'true' after 46.999 seconds and property 'present' of the second task is set to 'true' after 47.007 seconds. The task tags within the XML progress have their own 'id' and have a

reference to their corresponding content element in the XML content, namely attribute 'ref-content-id'.

3.3 Other platform processes

In addition to the authoring and playing processes there are also processes to monitor or support students, to manage game runs, to populate the platform and game runs and to integrate the platform with other systems.

Monitoring and supporting students. A tutor may (i) inspect a student's progress or (ii) interfere in a running game as an NPC or an administrator (iii) may adjust a student's progress. These sub-processes involve the Progress Manager or Shared Progress Manager (see section 3.2) to retrieve or adjust student's progress.

A tutor may have two reasons to inspect progress: to inspect the performance of a student or to help a student out if he does not succeed to finish a specific task. To inspect their performance he gets an overview of students per run, the tasks they have completed and assignment outcomes they have submitted. He may also generate his own overviews of specific game content properties, e.g., to inspect if certain resources are opened or questions are asked, or to inspect students' paths through the game. To help a student out he can open the player environment showing the student's progress to diagnose the problem. The same option allows him to interfere in a running game by sending an email as an NPC, e.g., in case a student's performance is poor or better than average. Sending such an email might also trigger script that releases a next task. Such thresholds placed by a game developer may enable a tutor to guarantee a certain level of quality.

As a tutor can help out a student with substantive problems an administrator can help out in case of technical problems, e.g., if a door stays locked or certain resources do not become available due to bugs. An administrator has an overview of all students within specific runs and can open the player environment showing the progress of a specific student to diagnose the problem. Subsequently he can fix the problem by adjusting property values within the student's progress. To adjust a property value he opens a similar dialogue as the one used to create a script action, which, after all, is meant to adjust a property value. If he saves the changed property value the Shared Progress Manager (see section 3.2) is used to send a p2p event to the student, which will result in a live update of the student's progress so he can continue his game session.

Managing game runs. The run manager has an overview of all his runs and can perform CRUD operations on them. Creating a run involves selecting a game from a list of all available games, giving the run a name and entering a start date and end date. It is important to stress that the game itself is not published but that the run just has a reference to the game. This allows for adjustments of the game that are immediately available to students, e.g., in case of required bug fixes. It is even possible to let stu-

dents start with a part of a game while another part still is in development. The latter option has been used on various occasions.
Just like the administrator the run manager has the possibility to adjust property values within students' progress during a run, but he may also initialize property values before a run. However, properties are not changed for one student but for all students in the run which allows for, e.g., assigning certain tasks to all students in the run at the same time. It also allows for configuring runs differently, e.g., in case of experiments where different runs should meet different experimental conditions. To adjust a property value the run manager uses a similar dialogue as the administrator (see above). If he saves the changed property value the Shared Progress Manager is used to send p2p events to all students in the run, which will result in a live update of students' progress.

Populating the platform and game runs with users. The administrator has an overview of all platform users and can perform CRUD operations on them. Creating a user involves assigning one or more platform roles to him, i.e., administrator, developer, run manager, tutor or student. Users can also be imported using an XML file containing predefined user data.

From his run overview the run manager can assign students to runs and game roles by selecting them directly or by importing them using an XML file containing predefined run user data. In the same overview he can also assign tutors to runs. He may also create teams of run users that have a shared progress for game components that allow user generated content such as the Resources and Google maps components. Students may extend the content of these components with their own resources and markers where extensions are stored in run team progress, which enables to see each other's contributions to a same game component.

Integrating other systems. The platform offers SOAP (W3C, 2017) web services that support the exchange of progress data with other applications that may adapt the player environment for a certain student. In an experiment, another application used webcam data analysis to derive a student's mood and used the web services to trigger personalized game support depending on his mood (Bahreini et al., 2012). The web services allow for retrieving or setting state element property values of States components. If a property value has to be set the corresponding web service creates a property event for the student and invokes the Event Handler (see Figure 3.29) to handle it. Script conditions and actions then are used to adapt the game support depending on the set property value. Other web services support identity management to enable single sign-on.

In another experiment we integrated the Unity Web Player. A Unity game was shown embedded within the player environment by using a plugin element of the Navigation component. JavaScript functions were used to exchange data between the Unity game

and the EMERGO player environment, which resulted in adaptation of each other's interfaces.

4 The platform architecture

The requirements for the platform formulated in chapter 2 led to the choice for a multi-layered client-server architecture (see Figure 3.30), which fosters abstraction, loose coupling and separation of concerns (Microsoft, 2009). One layer's implementation may be substituted by another implementation without affecting the other layers, e.g., when switching to another DBMS (Database-Management System). This architecture forms the foundation for the implementation of the different platform components and processes. To foster technical platform independence we chose the Java EE platform (J2EE, 2017) for implementation and MySQL server for data storage (MySQL, 2017). We chose the Hibernate framework for data mapping (Hibernate, 2017), the Spring framework (Spring Framework, 2013) for data management, and the ZK framework (ZK Framework, 2013) for the user interface. To host the EMERGO platform we chose the Apache Tomcat web server (Apache Tomcat, 2017). The platform has been developed under the GNU General Public License and is available on SourceForge at https://sourceforge.net/projects/emergo/.

As every architecture layer in Figure 3.30 depends on layers underneath we will describe the four different layers and their main components and responsibilities from the bottom to the top.

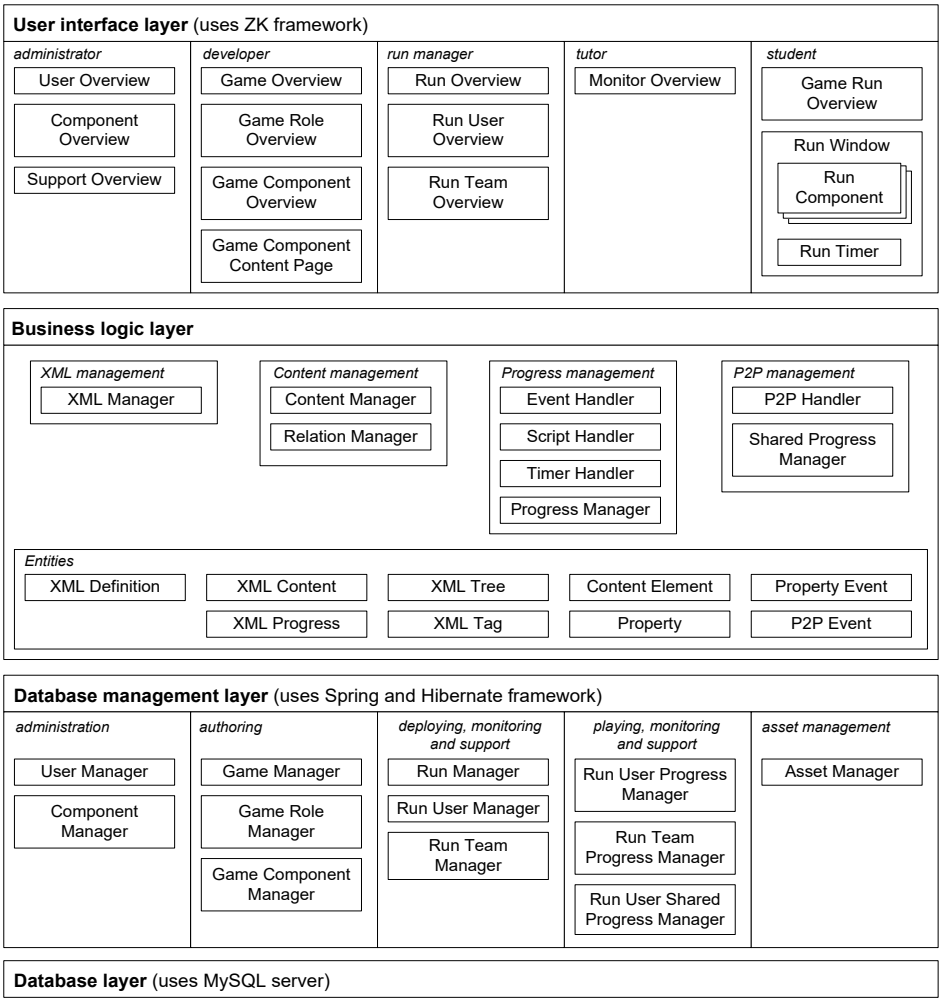

Figure 3.30. The platform's multilayered client-server architecture

4.1 The database layer

The platform stores all data in a MySQL relational database that is accessed using a JDBC driver. The database contains one table per domain model entity (see Figure 2.4) and some additional tables for the relations between these entities. The database also has a table to store asset entities that are uploaded by developers or students. These assets have no direct relation with the domain model entities in the database, but are referenced from within game component's XML content or XML progress. Another table is used for sharing students' XML progress.

4.2 The database management layer

In this layer every domain model entity has its own manager component. Five types of processes can be distinguished. Administrative processes involve platform users and components. Game authoring involves games, game roles and game components. Game deployment involves runs, run users and run teams. The same entities are involved in monitoring and supporting students when overviews of these entities are generated. Game playing involves run user progress, run team progress and run user shared progress. The same entities are involved in monitoring and supporting students when the player environment is used to inspect a student's progress. Asset management involves assets that are referenced from within game components' XML content or XML progress. All manager components offer CRUD (Create, Read, Update and Delete) operations on entities, and provide possibly filtered lists of entities to be used in the Business logic layer or User interface layer. The managers also validate entities so check if all entity attributes are entered correctly, except for entity attributes that contain XML. XML content is validated by the Content Manager in the Business logic layer and XML progress by the different Run Components in the User interface layer. The Spring framework is used to inject the manager components within the platform. All domain entities are implemented as Java Beans that have properties containing entity specific data or references to other entities that reflect their relations within the domain model. The managers use the Hibernate framework to map these Java Beans to database tables.

4.3 The business logic layer

This layer shows four parts that handle different business processes and one part with entities that play a role in one or more of these processes. We will describe the different processes starting with XML management, which is intensively used by the other three processes.

XML management. The XML Manager component is used to manage XML definitions of platform components, XML content of game components and XML progress of run user progress, run team progress and run user shared progress. It has six functions: (i) parse XML strings and return XML trees of XML tags (see below), (ii) convert XML trees back to XML strings for storage, (iii) offer CRUD and copy operations for XML tags, (iv) validate entered XML content using XML definitions, (v) provide filtered XML trees, e.g., on type of tag, and (vi) personalize an XML content tree with XML progress by extending every XML content tag with a reference to its corresponding XML progress tag. An XML tag component has properties like name, value and attributes just like a normal XML tag in a string and it has methods to get or set this kind of data. XML tags of type 'node' correspond to content elements that have content children, which contain their con-

tent, and properties (see section 2.1). The XML Tag component has methods to get and set the content element's content and properties.

Content management. The Content Manager is responsible for managing XML content, which includes rendering of the game component content page and the popup dialogue for editing content, and handling CRUD operations on XML content. It uses the XML Manager to validate entered content and the Game Component Manager in the Database management layer to retrieve or store the content. It also uses the Relation Manager to update the relations with other content. See section 3.1 for a detailed description of content management.

Progress management. The Event Handler is responsible for handling all property events which are events sent in case of change of property values. It uses the Progress Manager to store a changed property value and the Script Handler to possibly execute script related to the property change. If the script execution results in another changed property value the Script Handler will send a property event to the Event Handler to handle the changed property value. The Event Handler forwards every property event to the Run Window in the User interface layer so it may update itself. The Timer Handler is responsible for handling script timers. If a script timer fires it sends a property event to the Event Handler. See section 3.2 for a detailed description of progress management and event handling.

P2P management. The P2P handler is responsible for handling all p2p events which are events that are sent in case changed property values have to be shared with other students, e.g., when an email or chat is sent. The P2P handler uses the Shared Progress Manager which will store the changed property value for other students. The P2P Handler regularly checks for p2p events from other students, converts them to property events and sends them to the Event Handler to handle the changed property value. See section 3.2 for a detailed description of P2P management.

4.4 The user interface layer

This layer uses the Ajax-based ZK framework to expose the EMERGO platform to its users and to handle creation and user events. Every component in this layer represents a so called ZUL (ZK User interface Language) page on the web server that is rendered as an HTML page in the browser. This layer is separated into five parts that relate to the different platform roles. All 'Overview' components are used to manage or inspect domain model entities and use the Database management layer to present lists of entities that allow CRUD operations on them if applicable. The Support Overview and Monitor Overview also use the Run Window for inspection of student's progress.

Administrator. The User and Component Overviews present lists of current users and components and allow for CRUD and copy operations on them. Users can also be imported using an XML file containing predefined user data. A component's XML definition

is edited in an input field, so involves no specific editor. Components can be exported and imported as an IMS Content Package (IMSCP-IM, 2008) for deployment on other platform instances. The Support Overview is used to help students who are technically stuck in a game. See section 3.3 for a detailed description.

Developer. The Game, Game Role and Game Component Overviews present lists of a developer's current games, and game roles and game components per game and allow for CRUD and copy operations on them. A game or game component can be exported and imported as an IMS Content Package for deployment on other platform instances. The Game Component Content Page uses the Content Manager in the Business logic layer to handle entering of content for all game components. See section 3.1 for a detailed description.

Run manager. The Run, Run User and Run Team Overviews present lists of a run manager's current runs and run users and run teams per run and allow for CRUD operations on them. Run users can also be imported using an XML file containing predefined run user data. See section 3.3 for a detailed description.

Tutor. The Monitor Overview is used to monitor students' progress. It shows a list of runs a tutor has been assigned to by a run manager and allows for inspecting a specific student's progress by opening the player environment showing the student's progress. See section 3.3 for a detailed description.

Student. The Game Run Overview shows a list of runs a student has been assigned to by a run manager. If a student clicks on a run, the player environment is opened showing the Run Window. The Run Window functions as the stage for Run Components that represent game components created and authored by a developer. Run Window and Run Components are rendered using the game skin selected by a developer. Run Components will convert user events into property events and will invoke the Event Handler in the Business logic layer to handle them. A Run Timer is used to regularly check script timers by invoking the Timer Handler and to regularly check for p2p events from other students by invoking the P2P Handler. See section 3.2 for a detailed description of the structure and working of the player environment's components.

Chapter 4

Evaluating the usability of authoring environments for serious games

This chapter has been published as: Slootmaker, A., Hummel, H. G. K., & Koper, E. J. R. (2017). Evaluating the usability of authoring environments for serious games. *Simulation & Gaming, 48*(4), 553-578

Abstract

Background. The EMERGO method and online platform enable the development and delivery of scenario-based serious games that foster students to acquire professional competence. One of the main goals of the platform is to provide a user-friendly authoring environment for creating virtual environments where students can perform authentic tasks.

Aim. We present the findings of an in-depth qualitative case study of the platform´s authoring environment and compare our findings on usability with those found for comparable environments in literature.

Method. We carried out semi-structured interviews, with two experienced game developers who have authored a game for higher education, and a literature review of comparable environments.

Findings. The analysis shows that the usability of the authoring environment is problematic, especially regarding understandability and learnability, which is in line with findings of comparable environments. Other findings are that authoring is well integrated with the EMERGO method and that functionality and reliability of the authoring environment are valued.

Practical implications. The lessons learned are presented in the form of general guidelines to improve the understandability and learnability of authoring environments for serious games.

1 Introduction

Serious games (SGs) are considered to provide powerful and attractive ways of acquiring professional competences. However, their use is still limited, because their technical requirements are high, they are: difficult to customize for the educational process, difficult to support, and, the field lacks standards for SG design (Klemke et al., 2015). In addition, the field also lacks good architecture for SG development (Nadolski, Hummel, Slootmaker, & Van der Vegt, 2012) and simpler authoring tools (Arnab et al., 2012). In the previous decade, our the Open University of the Netherlands experienced similar needs and developed their EMERGO method (in English EMERGE: Efficient Method for ExpeRiential Game-based Education), including an online platform for SG development (Nadolski et al., 2008). It intends to both simplify and better support the development and delivery of scenario-based SGs. In this kind of game, students acquire professional competences in complex problem spaces that mimic real-world situations. The scenario describes the problem space and how it should adapt to the students' actions. The platform's authoring environment is used to convert the scenario into platform content.

Evaluations of the platform (Nadolski et al., 2008; Slootmaker, Kurvers, Hummel, & Koper, 2014) show that educators, after having received some instruction, can author most platform components independently and with ease. However, the authoring environment has not yet been evaluated in detail. We believe a deeper understanding is relevant, for both practice and research.

This type of research is part of the field of game design research, as Kultima (2015) examines and discusses it, stating that understanding game studies as design research could deepen our understanding of game design.

This article presents the findings of an in-depth qualitative case study on the usability of the authoring environment and its integration with the EMERGO method. We compare the usability findings with those found in literature for comparable environments. We present the lessons learned in the form of general guidelines to improve the understandability and learnability of authoring environments for SGs.

We first give some information on game design research, game authoring, usability, and comparable studies in the section *Background*. In the *EMERGO* section, we present the EMERGO method and the development, authoring, and debriefing of EMERGO games. In the *Method* section, we explain the method that is followed in order to arrive at our findings in the *Findings* section, in which we also provide practical guidelines for improving the understandability and learnability of authoring environments for serious games. Finally, in the *Conclusion and discussion* section, we present the main conclusions to be drawn from this study.

2 Background

2.1 Game design research

Although *game design* is the most popular keyword in game research papers, there is no explicit reflection on notions of *design* and *design research* (Kultima, 2015). In addition, game studies have focused on the *game* and the *player* but not on the *context* that involves the *design*, *designer*, *process*, and *practice* of the game. According to Kultima (2015), trying to understand game studies as design research would potentially improve our understanding of game design and bridge the epistemic gap between practice and science. She therefore encourages taking design research as theoretical background for future game studies. For possible theoretical background to be utilized, the author refers to Cross (2007), who distinguishes three areas of design research that are based on, respectively, people, process, and products: design epistemology (study of designers' professional theories), design praxeology (study of the practices and processes of design), and design phenomenology (study of the form and configuration of artifacts). Kultima (2015) argues that the multidisciplinary domain of game studies has been unsuccessful in addressing the latter two areas of design research.

To facilitate future studies and understanding of game design practice, Kultima and Sandovar (2016) proposed a framework of game design values. Their framework utilizes three design theory frameworks from architectural and industrial design: Lawson's Guiding Principles, Schön's Appreciative Systems, and Holm's Designers' Distinctive Design Values. All three frameworks indicate that designers never start from scratch but already have their own motivations, their own reasons for wanting to design, and their own sets of beliefs, values, and attitudes. The author divides game design values into nine categories: (1) Value of Player Centrism, (2) Casual Game Design Values, (3) Traditional Game Design Values, (4) Societal Impact and Cultural Values, (5) Value of Artistic Expression, Innovation, and Experimentation, (6) Values of Production and Creation Process, (7) Ludological Values, (8) Values of Independency, and (9) Commercial Values. The first three categories are more oriented toward players and involve values like usability and playability, flexibility and simplicity, and immersion, challenge, and competition. The fourth category is oriented toward society and culture and involves values like ethics and morality and cultural diversity and tradition. Categories five through eight are more oriented toward game developers and involve values like visual design and aesthetics, development as a challenge, collaboration, value of teamwork, value of game mechanics, and autonomy and artistic freedom. The last category is oriented toward business and involves values like economic success. The author states that, if viewed from a general perspective, there is no single design value that is more important than another. However, this case-based research focuses on game developers and more specifically on the game authoring process that can be classified under the category Values of Production and Creation Process.

2.2 Authoring leisure games

Current video games are often complex, immersive games that are developed through large and costly projects and involve many specialized developers who use specific development tools. The flexibility, productivity, and usability of these dedicated tools are decisive success factors. Game designs are complex when they entail many game elements to be classified under four categories: story (narrative), aesthetics (look and feel), technology (materials and interactions), and mechanics (fostering game rules and interactivity) (Schell, 2008).

The development of complex games requires comprehensive and dedicated development tools. Hartson and Pyla (2012) identified *two types of system complexity* that may influence usability: (1) *interaction complexity*, which is related to the intricacy or elaborateness of user actions, including cognitive load; and (2) *work domain complexity,* which is related to the degree of intricacy and the technical nature of the corresponding field of work (e.g., game development). Systems with high interaction and high work domain complexity are more likely to have low usability. This is in line with Oja (2010), who stated that usability is even more critical for complex software development.

No single definition for usability currently exists. Nielsen (1994) defined usability by its quality of five components: (1) learnability (for novice users), (2) efficiency (amount of time to accomplish task), (3) memorability (for frequent users), (4) errors (number, severity, recoverability), and (5) satisfaction (pleasantness). ISO/IEC (2011) defined usability as the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. A more recent concept is user experience (UX), which involves the effects of usability factors, usefulness factors (how useful a tool is for a task), and emotional impact (broader than Nielsen's satisfaction) and strongly depends on the context of a usage by a particular user (Hartson & Pyla, 2012). Not all UX aspects are equally important for every type of user. Paakkanen (2014) finds that experienced video game developers find effectiveness (usefulness) most important. For novice users, this probably would be the ease of use (usability).

Popular video game development tools, like Unity3D, Unreal Engine 4, and Cry Engine, have a high interaction and work domain complexity. They include many sophisticated editors (e.g., AI editors) that all contribute to their complexity and steep learning curve. Pattrasitidecha (2014) found that, out of 16 3D mobile game engines, Unity3D is easiest to learn but has a relatively low usability, which most likely is due to Unity3D's high complexity.

Popular console and web-based editors, like Super Mario Maker and Scratch, have low interaction and work domain complexity. Super Mario Maker is a very user-friendly editor for the Wii U console. Its interaction complexity is low, because games can be

easily created by dragging and dropping objects from a tool palette. Objects can even be combined to get a new object that shows combined behavior. Its work domain complexity is also low, because the number of objects and possible manipulations on it are limited, and objects have built-in behavior, so no entry of game rules is needed. This low complexity contributes to its high usability. Scratch is a massively used, low threshold web-based game editor. Its interaction complexity is low, because it makes use of blocks that can be dragged and dropped and connected to each other to create the game flow and even to create the game rules. Its work domain complexity is also low, because the number of different blocks is limited. Again, low complexity contributes to high usability.

Besides interaction and work domain complexity, design decisions may also influence tool usability. Murray (1999, 2004, 2016) investigated design trade-offs for authoring tools. Increasing the flexibility (the ability to author a diversity of game types), breadth (of the domains supported), or depth (of the models to author) of a tool usually comes at the cost of usability. In addition, learnability and productivity are often in conflict, because simplicity for novice users means less powerful features that foster productivity for experienced users.

Conditions during development and implementation of a tool may also influence its usability. What is the budget? What is the time schedule? Are there great risks involved? Are the right people with the right skills available? What is the expected number of authors? In case of setbacks or a small number of authors, the priority will likely be to deliver a tool that works, so usability aspects, efficiency, and satisfaction will be at the expense of effectiveness.

2.3 Authoring serious games

Unlike leisure games, serious games should support learning. Dede (2009), Clark and Mayer (2011), and Thillainathan and Leimeister (2014) stated that the learner should be in control and that learning should be situated and authentic, possibly based on a didactical model or approach and should support transfer of learned skills. To be able to give relevant support, guidance, and feedback, the game should keep progress of and assess the learner and should adapt to the learner's learning strategies and skills. The game may assess the learner on performance, emotion, motivation, and on personality aspects and may adapt to the learner on the micro and macro levels (Kickmeier-Rust & Albert, 2010; Kickmeier-Rust & Albert, 2012; Kickmeier-Rust, Mattheiss, Steiner, & Albert, 2011). Micro-level adaptation is embedded in the game flow and leads to, e.g., giving the learner advice or feedback or motivating or urging him. Macro-level adaptation leads to adjustments of either the game flow or the game's pace or intensity.

Murray (2004, 2016) did some important work on the *complexity of authoring tools* and stated that the complexity level of an authoring tool should match the complexity ca-

pacity of its user. He identified four types of complexity. *Interface and tool complexity* is related to the number of editor features and components. The more features and components, the more difficult it is to manage and combine them. *Object complexity* is related to the number of abstract concepts whose definitions and uses are not obvious. For instance, it is more difficult to understand and explain an abstract concept like feedback, when compared to a more concrete object like an image. *Structural complexity* is related to the number of complex structures of linked objects. Creating and maintaining such structures is cognitively challenging. *Dynamic complexity* is related to the number of laws, rules, mechanisms, or influences that contribute to change, which may lead to many possible student paths that are difficult to test and debug.

In addition, Murray (2004, 2016) identifies *five possible types of users* with *different complexity capacity*. *Teachers* have a low complexity capacity, so they cannot be expected to use complex authoring tools. This is in line with Theodosiou and Karasavvidis (2015), who found that student teachers struggle to incorporate critical game elements and have major difficulties in connecting game elements effectively. *Domain experts and content developers* have a medium complexity capacity, though they may have little practical or theoretical knowledge of pedagogy. *Instructional designers and learning theorists* have a medium complexity capacity too, though they may not have the time to dedicate to a steep tool learning curve. *Knowledge engineers and game developers* have a medium to high complexity capacity, because they are trained for representing knowledge in a computationally usable fashion. *Computer scientists and software developers* have a high complexity capacity, because they are used to design and debug structural and procedural models. Only the last two types of users can be expected to manage sophisticated authoring tasks.

2.4 Related work

Other authors also evaluated the usability of authoring environments for SGs. Mehm, Göbel, and Steinmetz (2012) evaluated STORYTEC, an authoring tool that integrates the work of game designers, pedagogues, artists, and domain experts into one unified authoring tool. Most usability aspects were found to be average. Only so-called *self-descriptiveness* (the dialogue should make clear what the user should do next) (ISO, 2006), which relates to understandability, and error tolerance, which relates to operability, were rated lower than average.

Van Est, Poelman, and Bidarra (2011) evaluated SHAI, a (prototypical) scenario editor for simulation games that enables instructors to arrange scenario building blocks to match individual trainees' needs and to make real-time adjustments. Usability was found to be poor. Shortcomings relate to understandability ('options are too complex', 'graphics are unclear', 'large scenarios are difficult to comprehend'), learnability ('better

descriptions are needed'), operability ('keyboard shortcuts are missing'), and user interface aesthetics ('better presentation is needed').

Marchiori et al. (2012) evaluated WEEV, a method and system for educational adventure game authoring, and identified many usability problems related to understandability ('part of the system is complex to use', 'example games are needed to understand the purpose of the system') and learnability ('a guided tutorial is needed to help novice users').

Gaeta et al. (2014) evaluated an authoring tool for the creation of stories that support learning in an emergency context and found usability to be relatively low.

3 EMERGO

We developed the EMERGO method and online platform (Nadolski et al., 2008) to simplify and better support the development and delivery of scenario-based SGs. In this kind of game, learners are confronted with realistic, ill-defined problems, often allowing multiple solutions and requiring application of necessary methodologies or tools and collaboration with fellow learners (Westera, Nadolski, Hummel, & Wopereis, 2008). The platform's authoring environment offers 22 components that support different (didactical) functions that should be present in scenario-based SGs. In addition, the platform offers environments to play the developed games, to monitor students, and to manage users and game runs. EMERGO has been used to develop 24 games for all kinds of disciplines. It supports the acquisition of four out of five capability types, as defined by Gagné (1985): intellectual skills, cognitive strategies, verbal information, and attitudes. Motor skills are not (yet) supported. The online platform is Open source and is available on SourceForge (EMERGO, 2013).

Earlier and more superficial evaluations of the authoring environment show that educators, after receiving some instruction, could use most components independently and with ease (Nadolski et al., 2008; Slootmaker, Kurvers, Hummel, & Koper, 2014). One component could not be used independently, and two components were not easy to use. This makes us question why the usability of some components is lower than of others, how we could improve this, how this is related to component complexity, and if it is related to the conversion of the scenario into game content. The research goal of this case study is to evaluate in detail the usability of the EMERGO authoring environment and integration of authoring with the EMERGO method. We expect that the usability evaluation will enable us to derive some guidelines for increasing the usability of the environment.

This case study addresses two areas of design research that are still underrepresented in game studies (Kultima, 2015): design praxeology (i.e., the development process using the EMERGO method) and design phenomenology (i.e., the EMERGO authoring environment). The study deals with the game design value category *Values of Production*

and Creation Process, as proposed by Kultima and Sandovar (2016), which contains values like *Technological advancement*, *Development as a challenge*, *Collaboration and value of teamwork*, and *Open source ideology*.

In this article, we use usability as defined by ISO/IEC 25010:2011 (ISO/IEC, 2011). It is one out of eight software quality characteristics and is defined as the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. The other characteristics are functionality, reliability, performance efficiency, compatibility, security, maintainability, and portability. Usability is further subdivided into six aspects: understandability, learnability, operability, user error protection, user interface aesthetics, and accessibility. Note that, for better readability, we replace the ISO/IEC characteristic 'functional suitability' with functionality and the characteristic 'appropriateness recognizability' with understandability.

In the following sections, we will describe development, authoring, and debriefing for EMERGO games and the EMERGO authoring environment itself.

3.1 Developing EMERGO games

The use of the EMERGO authoring environment is embedded in the EMERGO method. This method comprises five phases and (although based on ADDIE) recommends using iterations, like a Unified Process approach with cycles that prevent overspending and minimizes risks or failures.

During the analysis phase, the development team formulates answers to a list of standard questions. Answers are used as input for a global description of the game that includes learning goals and competencies to achieve. During the design phase, the method supports the team in the creative process of writing a scenario in three steps. First, the team formulates which activities have to be accomplished, why, when, where, and in what order, if needed. Activities are formulated as location plans, using the template *Where the student will…<description of the activity>*. Second, the team identifies: (i) with whom activities must be done and with what materials and tooling, (ii) when activities are completed and how this is assessed, and (iii) which feedback is given and when, and in what form and by whom. Third, the team describes each activity exhaustively in terms of its required materials and tooling. In this stage, it becomes evident whether materials are already available or still need to be developed and whether the scenario can be realized with available platform components, or if it needs new components or even a new game skin. During the development phase, the authoring environment is used to convert the scenario and materials into game content. If needed, new components or skin are developed, film recordings are made, and other materials like documents or images are developed. During the implementation phase, the game is de-

ployed to students and educators (for monitoring), and during the evaluation phase, the game is evaluated.

An EMERGO development team consists of content matter experts, educational technologists, interaction designers, and ICT developers. Educational technologists and content matter experts write the global description and scenario of the game and involve other team members to check for feasibility. Interaction designers and ICT developers develop graphical assets and, if needed, new components or skin. If film recordings are needed, the team temporarily is reinforced with cameramen, actors or experts, and video editors. Initially, our objective was for educational technologists and content matter experts to do all authoring, but actual practice shows that game script authoring is mostly too complicated and is then done by an ICT developer, which is in line with Murray (2016), who states that the latter user type has a higher complexity capacity than the former.

3.2 Authoring EMERGO games

In a typical EMERGO game, the student acts as a PC (Playing Character) and enters an authentic environment, where he works as a trainee. He can navigate to different locations, where he finds NPCs (Non-Playing Characters), like his supervisor, colleagues, experts, or specialists, or can attend interviews or meetings (see Figure 4.1). In the environment, he has a tablet with apps, e.g., a task overview, a resources app, an (in-game) email app, or an app to conduct tests. He also has a memo recorder, to record interesting parts of interviews and meetings, and a notepad to make contextualized notes.

The student gets tasks from his supervisor or other NPCs, either in person or by mail. He can be assessed on every action he performs, e.g., which interviews he attends, which questions he asks, which resources he consults, or which mail he sends. In addition, he can be assessed, e.g., using tests that enable measuring foreknowledge and performance. Depending on his actions, the game can adapt the environment at the micro level, e.g., by sending mail, showing an alert, changing an NPC reaction, releasing new resources or new interview questions, or changing an answer to a question, or at the macro level, e.g., by providing new or alternative tasks. The student gets feedback on his performance by NPCs in person, by mail, as screen text, or in tests. This feedback can incorporate mail attachments or the release of resources, such as worked-out examples or expert reports. The student gets navigation support through alerts, e.g., reminders for meetings or instructions for where to go next.

Figure 4.1. A patient being interviewed (video). At the bottom left, we see the tablet, and at the bottom right, we see the memo recorder and notepad

The EMERGO authoring environment offers 22 components to realize the above kind of scenarios (see Table 4.1).

The components support eight different (didactical) functions: present and adapt environment (E), assign tasks and provide overview (T), present knowledge (K), assess learner (A), provide feedback (F), support processing of information (P), support collaboration (C), and support navigation (N). Note that one component may serve several functions and that one function may involve several components. For instance, the *conversations* component can be used to assign a task, to present knowledge, or to provide feedback. And the *script* component should assess the learner to trigger the *conversations* component to give the right feedback.

The components allow much freedom in the way the environment is presented, how tasks are assigned, if they must be executed in a certain order, how they are assessed, and how feedback is provided and thus support a wide range of game scenarios. Of course, the metaphor of an environment with locations and the available components put constraints on the end form of the game, but this partially can be overcome by adding new components or a new game skin.

Table 4.1: EMERGO components and their possible functions and complexity

Component	Description	Functions	Complexity
Navigation	Enable spatial navigation through the game	E	Medium
Conversations	Enable communication with NPCs using video or text	ETKF	Medium
Notepad	Enable making contextualized notes	EP	Low
Memo recorder	Enable recording of conversations	EP	Low
Alerts	Provide popup texts	EFN	Low
Notifications	Provide (accumulated) embedded texts	EFN	Low
Scores	Provide score overview	EF	Low
Profile	Enable sharing profile with PCs	EC	Low
Chat	Enable communication with PCs	EC	Low
Tablet	Enable selecting apps	E	Low
Tasks	Provide task (completion) overview. App	ET	Low
Resources	Enable consulting resources. App	EKF	Low
Email	Enable communication with NPCs and between PCs. App	ETKFC	Medium
Assessments	Enable conducting tests. App	EAF	Medium
Logbook	Provide overview of notes. App	EP	Low
Memo player	Enable playing back of recordings. App	EP	Low
Google maps	Enable inspecting maps with markers. App	EK	Low
Directing	Enable analyzing communication between NPCs. App	EP	Low
Game manual	Provide help on game interface. App	EN	Low
Items	Define questions to be used in tests	EAF	Medium
States	Define states to be used by the Script or Scores component	A	Low
Script	Define rules to assess the learner and adapt the game at the micro and macro levels	ETKAFPCN	High

Based on Murray's (2004, 2016) four tool-complexity types, we identified three complexity levels: low, medium, and high. This allowed us to relate components' complexity to the complexity capacity of its user (Murray, 2016). Low complexity components have a low interface, object, and structural and dynamic complexity. They either have only a few configuration options or an obvious and simple data model without dynamics. Medium complexity components have a medium interface, object, and/or structural complexity, but a low dynamic complexity. The *navigation*, *conversations*, *email*, *assessments*, and *items* components comply with this condition. They all have a medium object complexity, because they include abstract concepts whose definitions and uses are not obvious. In addition, the *navigation* component has a medium interface complexity, because it includes a larger number of authoring features, and the *navigation*, *conversations*, and *items* components have a medium structural complexity, because they include complex structures of linked objects. High complexity components have a medium or high interface, object, structural, and dynamic complexity. Only one component, the *script* component, complies with this condition. It has a medium interface and a high object complexity, and depending on the number of game rules and their interrelations in the game scenario, it may have a high structural and/or dynamic complexity.

Depending on their individual complexity capacity, content matter experts will use low or medium complexity components that involve knowledge presentation. Educational technologists will mostly use medium complexity components that involve task assignment, assessment, and feedback. ICT developers will use high complexity components, although some educational technologists may also consider using them.

Object, structural, and dynamic complexities are related to the SG domain and therefore are difficult to influence. However, interface complexity is related to the way user tasks are translated into usable interfaces and therefore leaves room for improvement, which is the motive for our usability evaluation.

3.3 Debriefing EMERGO games

The EMERGO platform supports the design of in-game and post-game debriefing for the reflection on and the sharing of the game experience to turn it into learning (Crookall, 2010).

For the design of in-game reflection on the game experience, game authors can use components that support giving feedback or processing of information. The *conversations* component can be used for reflection on a task with a supervisor or reflection on the domain with an expert. The *resources* component can be used to provide additional reflection materials. The *email* component can be used for asking the student to send in a reflection document like a report or for commenting on a student's task process or outcomes by NPCs or PCs (fellow students or educators), including attachments like expert outcomes that can be used for reflection. If an educator has a PC role, he can support or moderate students' reflection during the game. He can even do this by impersonating an NPC. The *assessments* component can be used to reflect on learning, the *logbook* to reflect on notes made, and the *memo player* and *directing* component to reflect on NPC communication, e.g., a patient interview.

For the design of in-game sharing of the game experience, game authors can use the *email* and *chat* components that support communication with fellow students or educators.

For post-game reflection on and the sharing of the game experience, educators can organize debriefing sessions with students where all student data can be used as input to foster discussion. Educators can use the platform or ask administrators to provide overviews of students' performance and to reflect on developed games.

3.4 The EMERGO authoring environment

One of our main goals was to develop a user-friendly, reliable, and stable authoring environment that would enable efficient development of scenario-based SGs by offering a set of common components. We set up functional and non-functional requirements (see Appendix 2) that laid the foundation for the structure and the working of the environment, which comprises four pages (see Figure 4.2).

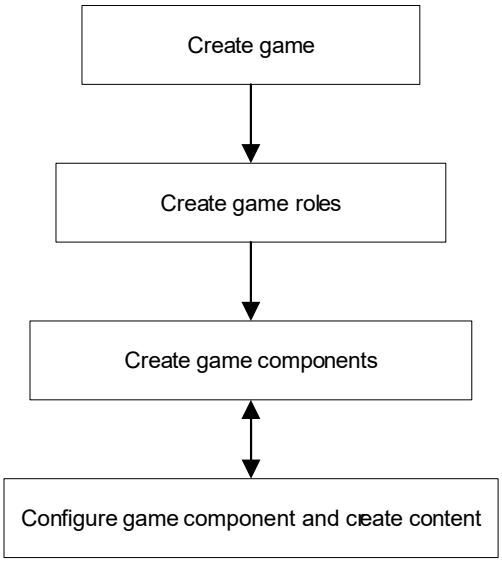

Figure 4.2. Authoring process

The *games* page shows an overview of games and allows CRUD (Create, Read, Update, Delete) operations on and import and export of games.

The *game roles* page shows an overview of game roles for a chosen game and allows CRUD operations on game roles. A game role can be either a PC or an NPC.

The *game components* page shows an overview of game components for a chosen game and allows CRUD operations on and import, export, and copying of game components. Most components allow for having multiple game component instantiations, which enables thematically arranging content, e.g., one *conversations* component per interviewee.

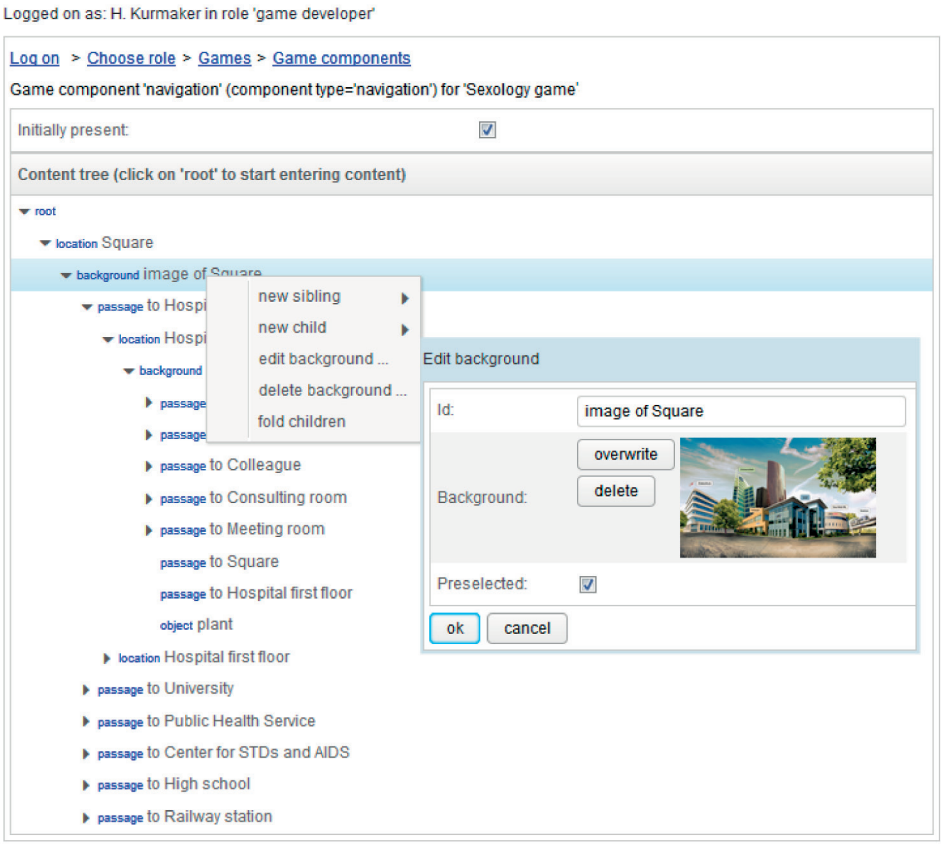

Figure 4.3. Game component content page for the Navigation component

The *game component content* page (see Figure 4.3) shows the game component configuration and content and allows CRUD operations on and drag, drop, and copying of game content. Configuration implies setting initial properties, e.g., is the game component initially present? Content is presented as a tree and authored using a single game component content editor whose working is determined by the component definition that defines possible content elements (e.g., locations or backgrounds), their hierarchy, which content can be entered, and their properties. A content element is edited in a pop-up dialogue that validates entered content. The *script* component is authored the same way. Script conditions are added as root tree items and will be triggered by property changes that are initiated by student actions, timers, or script actions. Script actions are children of a script condition and are executed if the condition is triggered. They change properties that may adapt the player environment or the game script itself.

To enter all content, authors will switch between the *game components* page and the *game component content* page. Game and game components can be previewed and tested in the player environment.

4 Method

As subject for our evaluation, we choose the development of a game on Sexology, one of the most recently developed games. It is a typical example of an EMERGO game, and it uses most EMERGO components, 14 out of 22 available.

4.1 Participants

Two experienced male game developers, who developed several other EMERGO games before, completed the game authoring. The first author is an educational technologist (without any technical background), who also led the project and wrote the scenario. The second author is an ICT developer, who also developed new game components. The educational technologist authored the *conversations*, *notepad*, *alerts*, *tablet*, *resources*, *email*, and *logbook* component. The ICT developer authored the *navigation*, *memo recorder*, *tasks*, *memo player*, *directing*, *game manual*, and *script* component.

4.2 Data collection method

As data collection method, we chose semi-structured interviews (Bryman, 2012). Strengths of this method are that it has a high validity, because interviewees are able to talk about something in detail and depth, and a high flexibility, because it allows complex questions and issues to be discussed. Weaknesses of this method are that it is not very reliable, because it is difficult to exactly repeat an interview, and that the findings are difficult to generalize. We found other data collection methods, like questionnaires, observation studies, think- or talk-aloud protocols, focus groups, automated collection of heat maps, or a combination of methods, not appropriate. Questionnaires, observation studies, and think- or talk-aloud protocols give too little detail and depth and are less flexible. Focus groups are more suitable for larger groups and bear the risk that opinions are not expressed equally. Automated collection of heat maps is not possible, because the EMERGO platform does not log the authoring process. A combination of methods is not appropriate, because the aforementioned methods are not appropriate.

We prepared the interviews by setting up an interview guide with themes and related questions (see Appendix 2). The themes are: (1) the author's general impression of the authoring environment, (2) the requirements for the environment, (3) the components used for the Sexology game, and (4) the development process.

4.3 Procedure

The same interviewer interviewed the two game authors separately. Both interviews were conducted about a year and a half after the game was developed, lasted about two and a half hours, and were recorded with consent. The spoken language was native, so interviewees could better express themselves. Interviewee and interviewer together walked through the pages of the authoring environment and the 14 EMERGO components to recall working with it in a natural setting.

For our data analysis, we first used the interview recordings to make notes per interviewee and per theme. Second, we identified issues and counted related remarks. Third, we related these issues to usability aspects and other ISO/IEC software quality characteristics and to aspects of the development process. As a last step, we collected suggested improvements to be able to set up general usability guidelines.

5 Findings

We present the findings related to our original evaluation goal, which was to evaluate the usability of the authoring environment and the integration of authoring in the EMERGO method. As interviewees also made remarks related to other software quality characteristics and the development process itself, we present these findings as well. A general finding is that interviewees' remarks do not contradict each other. We end with general usability guidelines for authoring environments for SGs.

5.1 Usability of the authoring environment

We give a general impression per interviewee composed of their striking literal remarks.

The educational technologist. 'If you see it for the first time, you think: 'What on Earth do I have to do here?' However, things fairly quickly become clear if you get some peer support, and are somewhat familiar in scenario writing. You don't have to be an ICT developer, but it is good to get an idea of the different layers you can distinguish in authoring, where switching properties on and off is close to programming. If you get deeper, it conceptually becomes more complicated. If you work with it somewhat longer, almost all components are a piece of cake, except scripting. Then it becomes Spartan, because it is not always intuitive.'

The ICT developer. 'You actually see a somewhat empty environment. It has a *new* button, but further you see little information, especially for someone who knows nothing about it. You get no location or context-specific help. You might build in that you can get some explanation on every screen, on the purpose, what you can do exactly, and how you might proceed. If you understand the editor, it works fine. However, entering game script requires concentration to prevent errors.'

Interviewees made remarks that can be related to three out of six usability aspects, namely *understandability*, *learnability,* and *operability*. No remarks can be related to user error protection, user interface aesthetics, or accessibility. Both interviewees identified most issues. ISO/IEC defines understandability as the degree to which users can recognize whether a product or system is appropriate for their needs, learnability as the degree to which a product or system can be used by specified users to achieve specified goals of learning to use the product or system with effectiveness, efficiency, freedom from risk, and satisfaction in a specified context of use, and operability as the degree to which a product or system has attributes that make it easy to operate and control.

The understandability of the authoring environment is problematic. Although interviewees find the distribution in pages and navigation through them obvious, they quite often find the used terminology unclear and not fitting their expectations (16 remarks) and find it unclear as to why, when, and where certain options are present, or why two options offer the same functionality (25 remarks). Less problematic is that interviewees miss examples of scenarios, games, and game components (three remarks).

The learnability of the authoring environment is problematic. Interviewees miss on-screen guidance and clear instruction on all pages and pop-up dialogues (17 remarks) and miss information on didactics and use of the components, their mutual dependencies, and the order of entering component content (10 remarks). For a large part, missing guidance, instruction, and information can be found in a comprehensive authoring manual, but it is partly outdated, and searching in a manual for the right help is laborious.

The operability of the authoring environment is somewhat problematic. Interviewees mention that available components are not filtered on the chosen game skin (one remark), file names cannot contain special characters (two remarks), objects cannot be easily positioned on the screen (one remark), and the preview option does not always function as expected (two remarks). However, interviewees like drag and drop and the uniform input control for URLs and files.

Interviewees made no remarks about the *notepad*, *memo recorder*, *tablet*, *logbook*, and *memo player* components, probably because these components only need some configuration and no authoring of game content.

We give an impression per interviewee and authored component that is composed of their striking literal remarks.

The educational technologist. 'It is a lot of work, but not really complicated. After some time, it is no longer a problem to work with. It cannot be made easier. You just have to copy/paste from Word and work very precisely.' (*Conversations*). 'No problem, I understood everything.' (*Alerts*). No problem at all. The preview option is indispensable.' (*Resources*). 'I had some trouble to understand the distinction between sent and received mails. I further had no problems.' (*Email*).

The ICT developer. 'Part of authoring does not seem logical or is unclear, unless you imagine it visually. However, for a large part authoring is straightforward, once you know how it functions and what it stands for.' (*Navigation*). 'No problems, pretty simple.' (*Tasks*). 'Explanation is missing.' (*Directing*). 'It is somewhat difficult to understand the distinction between the three types of resources. The rest is obvious.' (*Game manual*). 'It is not entirely clear what every property stands for and some menu options are unclear. You can implement different solutions for the same problem. This all made authoring more difficult, especially if I had not used the component for a while. In itself I found authoring convenient.' (*Script*).

5.2 The development process and the integration of authoring

Interviewees made remarks that can be related to the development team, the EMERGO design and development phase, and the transition between these two phases. The educational technologist identified most issues.

Both interviewees find the development team very important. The educational technologist states that you need a good role distribution ('What I did really suits my role.'), the right people ('The quality of the content matter expert is almost decisive.'), and a good organization of such a project.

Team members should complement each other and should consult each other to get the best result ('Together you find solutions that you would not find separately.'). During the authoring phase, both authors were in close contact, so they could efficiently work together to make things work and fix bugs.

For the design and development phase, the educational technologist states that he wrote the scenario without taking the available components into account very much. However, novice developers should know how to deal with the components beforehand, otherwise they get into trouble. If they are expected to author only one game, they should author only low complexity components.

Both interviewees have their opinions about the moment of transition between the design and development phase. The educational technologist states that authoring normally starts when the scenario is finished, but that you could start earlier if the storyline is clear and you know which components you need, at the risk of time-

consuming adjustments in case of scenario changes. In addition, the ICT developer states that it also depends on someone's preference. One person likes to write everything down, while the other likes to try things out early.

We do not identify any problems related to the integration of authoring in the EMERGO method. Interviewees are positive about the process of converting the scenario to game content and are satisfied with the efficient way of authoring together. The ICT developer could quite easily extract script conditions and actions from the scenario, because the scenario is structured in such a way that this is possible.

5.3 Other software quality characteristics

Interviewees made remarks that can be related to two out of seven other software quality characteristics, namely *functionality* and *reliability*. No remarks can be related to performance efficiency, compatibility, security, maintainability, and portability. The ICT developer identified most issues. ISO/IEC defines functionality as the degree to which a product or system provides functions that meet stated and implied needs when used under specified conditions and reliability as the degree to which a system, product, or component performs specified functions under specified conditions for a specified period.

The functionality of the authoring environment is valued. Both interviewees stated that the preview and test options are essential and indispensable. They find it convenient that you can easily change the content of a game in exploitation. The ICT developer is positive about the flexibility of the Script component.

The reliability of the authoring environment is valued. Interviewees find saving content to be reliable, although drag and drop sometimes results in partial loss of data.

5.4 General usability guidelines for authoring environments for serious games

Just as with the EMERGO authoring environment, other authoring environments for SGs seem to have usability problems related to understandability and learnability (see section *Related work*). To prevent such problems, we present general guidelines that are based on improvements suggested by the interviewees.

Guidelines to improve understandability:

- *Offer an intuitive user interface* so authors can more easily do what they want. An intuitive interface might be different for different kinds of authors. The educational technologist: 'It is not always intuitive. For students, we now have an interface, whereby they only have to think about what they want to do, not how to do it. Can we improve the usability for authors in the same way? Probably you should make different kinds of screens for different kinds of authors.'

- *Offer two levels of input*, basic for novices and advanced for experts, so novice authors are shielded from unnecessary complexity. The ICT developer: 'Better first present mandatory input controls and then optional input controls.'
- *Offer examples* of scenarios, games, and game components and how they relate to each other, so authors better understand what to do. The educational technologist: 'If I would be a novice developer, I would like to see some example scenarios that highlight where and when certain components are used.'
- *Offer a preview option* to preview entered content at any time, so authors better understand what they are doing. The educational technologist: 'The preview option helped me to understand the working of the authoring environment.'
- *Use clear terminology* fitting user expectations. The educational technologist: 'You have not yet managed to name the properties in a way that I understand exactly what will happen.'

Guidelines to improve learnability:

- *Offer clear instruction and wizards*, so authors are guided during the authoring process. The ICT developer: 'Context sensitive help is needed when entering game content. You also could use wizards.'
- *Offer information* on didactics and use of components, so novice authors can make a quick start. The ICT developer: 'You could give a short description why and how to use a component.'

6 Conclusions and discussion

Our research goal was to evaluate in detail the usability of the EMERGO authoring environment and integration of authoring in the EMERGO method for serious game development. The case-based research addresses two areas of design research, as distinguished by Cross (2007) – design praxeology and design phenomenology – and falls into the game design value category *Values of Production and Creation Process*, as proposed by Kultima and Sandovar (2016).

We found understandability and learnability of the authoring environment to be problematic and operability to be somewhat problematic. On the one hand, this is caused by a lack of guidance and support but on the other hand, it is caused by the complexity of the domain and the environment itself. The first problem originates from the initial development of the EMERGO platform, when the priority was to build a player environment for students without enough capacity available to invest in the usability of the authoring environment. The second problem is related to tool complexity (Murray, 2004, 2016). Complex learning requires complex scenarios that need a powerful environment with a lot of functionality and freedom, which may lead to lower usability. So power and flexibility of the authoring environment are both a strength and weakness.

We found authoring to be well integrated in the EMERGO method. However, our evaluation method did not include detecting the opposite, because we did not ask a question that specifically addressed this issue. Limitations of the EMERGO method might be that it only supports one type of game (scenario-based). However, if the scenario does not involve motor skills and can be realized with the available components, other types of games (like point-and-click adventure games) might also be supported. In addition, the method does not dictate but rather offers tooling to support design and development. The way the scenario is set up (using location plans) leaves plenty of room for creativity. Interviewees also do not mention any problems that can be related to the separation between design and development phases. This might be because they do not know better, but it also has to do with efficient development. Interviewees stated that if development starts when the scenario is not mature enough, time-consuming adjustments are needed in case of scenario changes. Of course, the authoring environment might be used in a more creative and agile manner, by skipping the scenario and switching between generating content, entering content, and previewing.

We found functionality and reliability of the environment to be valued.

Our findings do not include all usability aspects and all software quality characteristics. We think this is not due to our evaluation method but to the fact that interviewees did not mention related remarks, either because they had no problems with it or it was not relevant in their context of use. For instance, they will have had to deal with the aspects *user error protection* and *user interface aesthetics*, but they made no related remarks. And the characteristics *maintainability* and *portability* are not relevant in their context of use, because they relate to the EMERGO platform itself, not to developed games.

Besides the limited scope on certain usability aspects and software quality characteristics, our evaluation has other limitations. The data obtained are based on the development of only one game by only two authors who did not use all EMERGO components. However, we have strong indications that our findings are generic for the development of all EMERGO and similar complex learning games, because of the following reasons. First, the developed game contains a didactical scenario that is representative of a typical EMERGO game. Second, we focused on the development process by eliciting knowledge from informants that were strongly connected to it. Third, the facts that these authors / informants had already developed and authored EMERGO games before, come from different backgrounds (different world views), and have used other components as well, make it very probable that their remarks are generic for other EMERGO games and components as well. Fourth, our findings are in line with more superficial findings we collected with other authors in two previous studies (Nadolski et al., 2008; Slootmaker, Kurvers, Hummel, & Koper, 2014). Fifth, all components are authored by a single editor that uses common input controls, so components that are not evaluated are also indirectly partly evaluated. We do not claim that our findings are

applicable to development of serious games in general, especially when these do not contain specific references to learning features.

Our method might not be reliable, since we only interviewed two authors who may have given desired answers. However, they are the most experienced and most recent authors, and we think that they were honest, also because they criticized the authoring environment a lot. The fact that the interviews were conducted long after the game was developed is a clear limitation. However, the authors have still used the authoring environment after the game was developed, and walking through the environment helped them recall their memories and led to a very detailed narrative.

We are not able to generalize our findings to specific learning features, such as assessment and feedback, because we did not raise them during the interviews. Also, the authoring environment has no single components that deal with learning features, like other environments (Kickmeier-Rust & Albert, 2010), but instead requires the cooperation of several components to support learning.

We presented some general usability guidelines for authoring environments for serious games. These guidelines include providing examples and didactic advice that might direct authors in a particular style of game, which may reduce the chances of other creative solutions. However, novice authors need examples and advice, and good and varied examples might also feed creativity. Another risk is that authors might get overfed by all the instruction and information or even do not pay attention to it. However, this probably depends on the type of person. There are people who prefer guidance and others who prefer trial and error. All authors should be served.

Most guidelines seem obvious but can easily be neglected under time pressure or due to other causes. Further, the guidelines seem to be so general that they may also be applicable to other types of authoring environments.

As a follow-up of this evaluation, we plan to impose the usability guidelines on the EMERGO authoring environment. In a future study, we will evaluate the environment again to see if the guidelines indeed cater to better usability. It would be interesting to investigate if a more graphical or block-based interface would be an improvement.

Since the development of the game used for this evaluation, the EMERGO platform has been extended with new functionality. For instance, the platform now supports more adventure-like games, like the Playground Game developed by Westera, Slootmaker, and Kurvers (2014). We recently integrated the use of the webcam to record students who counsel virtual patients. These recordings are used for in-game peer feedback and are discussed in a post-game debriefing session. We are currently working on a game for an introductory course on Psychology. This game also will be used for research purposes. We will simplify the rollout of games to different experimental groups.

We have plans to integrate an external service to analyze quality of reports and to add components that support collaboration. We already developed two games that use online collaboration (Hummel et al., 2010; Hummel, Geerts, Slootmaker, Kuipers, & Westera, 2013). We will use this experience to add new components for rating, voting, and negotiation.

Acknowledgements

We would like to thank Henk van den Brink and Hub Kurvers of the Open University of the Netherlands for participating in the interviews.

Chapter 5

Evaluating the usability of player environments for serious games

This chapter is based on: Slootmaker, A., Nadolski, R. J., Kurvers, H. J., Hummel, H. G. K., & Koper, E. J. R. (2018). Evaluating the usability of player environments for serious games. Manuscript submitted for publication.

Abstract

The web-based EMERGO platform enables the development and delivery of scenario-based serious games (SGs) for acquiring complex cognitive skills in authentic professional settings. One of the main goals of the platform was to provide an intuitive and immersive player environment allowing students to perform authentic tasks. We present the results of a both quantitative and qualitative summative study on the player environment's usability. For the quantitative part we used pre- and post-test questionnaires, and for the qualitative part individual note taking during game sessions and group discussions after game sessions. The analysis shows that the overall usability of the player environment is "ok". Its operability is valued most, but understandability and user interface aesthetics are considered somewhat problematic. We argue why problems are probably caused by inadequate game preparation and students' expectation to get a more realistic and less restrictive environment. We present guidelines to improve the usability of player environments for SGs.

1 Introduction

Serious games (SGs) are powerful means to provide learning in a more attractive and challenging way, e.g., to learn complex cognitive skills in authentic professional settings. These skills involve using, transforming or supplementing available knowledge, and so called higher-order activities like problem solving, reasoning, thinking, assessing, and concluding.

The actual uptake of SGs is still hampered because their development is technically demanding and involves high costs and time investments. In addition, the field lacks a good architecture for SG development (Nadolski, Hummel, Slootmaker, & Van der Vegt, 2012), standards for SG design (Klemke et al., 2015) including usability design, and standardized ways of evaluation including usability evaluation. Measuring usability itself is complex as Lewis (2014, p. 664) emphasizes: "The measurement of usability is complex because usability is not a specific property of a person or thing. You cannot measure usability with a simple 'usability' thermometer (Dumas, 2003; Hertzum, 2009; Hornbæk, 2006). Rather, it is an emergent property dependent on interactions among users, products, tasks, and environments".

To overcome a number of problems related to SG development, we developed the EMERGO (in English EMERGE: Efficient Method for ExpeRiential Game-based Education) web-based platform for the development and delivery of scenario-based SGs (Nadolski et al., 2008). Scenario-based SGs foster knowledge acquisition, development of cognitive skills and understanding of complex relationships. The scenario describes the problem space, the learning tasks to be performed, how this is assessed and how the game should adapt to the student, e.g., by providing personalized feedback. The platform enables a wide variety of game scenarios to be authored, to be played and to be monitored, integrates game development, delivery and playing in one system, and can be used for education as well as for research purposes. Fast and flexible game development is fostered by options to reuse game content, to preview and test a game at any time, to adapt a game to the individual student and to easily modify already deployed games. Fast and flexible game delivery is fostered by web-based delivery of games, and of updates of games and the platform itself, and options to interfere in a running game and to deliver different game versions of the same game to different target groups. Over the years, the platform has been used by thousands of students and has been extended with many new platform components.

One of our main goals was to provide an intuitive and immersive player environment that enables students to perform authentic tasks. The research goal of this study is to now evaluate the usability of this player environment in detail and to establish guidelines to improve the usability of player environments for SGs. We already evaluated the platform's authoring environment in detail (Slootmaker, Hummel, & Koper, 2017). However, the player environment, although being essential for the platform, has not

been evaluated in detail until now. More superficial evaluations with less available platform components (Nadolski et al., 2008; Slootmaker, Kurvers, Hummel, & Koper, 2014) have shown that students are (very) satisfied about the user interface of the player environment. We remained interested to know whether students are still satisfied, about which usability aspects, whether and how platform components differ in their perceived usability and whether we could derive usability guidelines for this type of environments. We believe a deeper understanding is relevant to both the EMERGO player environment and comparable player environments for SGs.

We will now first give background information on the concept and definition of usability (in the 'Usability' section). In the 'EMERGO' section we then present EMERGO, its player environment and the available platform components under study. In the 'Method' section we explain the methods for data collection and analysis we applied to arrive at our results and findings, including practical guidelines for player environments for SGs (in the 'Results and findings' section). Finally, in the 'Conclusions and discussion' section, we present our main conclusions to be drawn from this study.

2 Usability

Although usability is a very important quality factor of a software system, no single definition of usability exists which takes into account all of its possible aspects (Dubey & Rana, 2010). Nielsen (1994), for instance, defined usability by its quality of five components: learnability (for novice users); efficiency (amount of time to accomplish task); memorability (for frequent users); errors (number, severity, recoverability); and satisfaction (pleasantness). ISO/IEC (2011) on the other hand, defined usability as the degree to which a product or system can be used by specified users to achieve specified goals with effectiveness, efficiency, and satisfaction in a specified context of use. A more recent concept is "user experience" which involves the effects of usability factors, usefulness factors (how useful is a tool for a task), and emotional impact (Hartson & Pyla, 2012).

Apart from a lack of a single usability definition, there is also no single method to measure it. The most widely used standardized usability questionnaires are the QUIS (Chin et al., 1988), the SUMI (Kirakowski, 1996), the PSSUQ (Lewis, 1995, 2002) and the SUS (System Usability Scale; Brooke, 1996, 2013; Sauro, 2011). The SUS consists of 10 items, has a global reliability of 0.92 and is probably most widely used (Lewis, 2014). It produces one usability score, allows for interpretation of its data in a normative way, and seems to tolerate minor changes to its wording (Lewis, 2014) and translations into other languages (Sauro, 2011). A disadvantage of these questionnaires is that they either produce a general score (SUS) or scores on general usability aspects (QUIS, SUMI and PSSUQ), which makes them less appropriate to identify more specific interface related usability issues.

Usability is a decisive success factor for video games, which is illustrated by the large volume of studies on the usability and playability of games. The concept of "playability" is broader than that of usability and is defined as "the degree to which a game is fun to play and is usable, with an emphasis on the interaction style and plot-quality of the game; the quality of gameplay" (Usability-First, 2017). Playability may be affected by the quality of the storyline, the degree of responsiveness, the intensity of interaction, pace, control, intricacy, customizability, realism, social and team support, and the quality of graphics and sound. Federoff (2002) compiled a list of game usability heuristics that can be used for video game creation and evaluation, and classified them into three areas: game interface, game mechanics (fostering game rules and interactivity), and gameplay (problems and challenges a player must face). The game interface and game mechanics areas cover usability aspects like interface consistency, while the gameplay issues cover more typical playability aspects like a variable difficulty level.

For SGs the same usability and playability aspects play a role as for video games. However, because their main purpose is learning (Franzwa et al., 2014), aspects that support learning obviously need more attention. According to Ibrahim et al. (2012), SGs aim to motivate learners, give them appropriate feedback, improve their skills at the right level, and improve collaboration within groups. The author compiled a list of playability guidelines to evaluate and enhance the playability of SGs, which mainly fall into Federoff's gameplay area (2002) and cover all SG aspects, e.g., game challenge (like "not too difficult" or "easy"); feedback (e.g., to understand why one has failed); adaptation (e.g., to the individual pace of the player); and game control (e.g., a player should be in control). According to Hamari et al. (2016), both engagement and challenge in game-based learning have a positive effect on learning.

Player environment for SGs will be partly responsible for the usability and playability of developed games. Game usability will mainly depend on the usability of the environment's different components. However, game playability will mainly depend on the quality of the game itself, e.g., the quality of the storyline, feedback, graphics and sound, which cannot be influenced by player environments. However, player environments should support playability aspects like responsiveness and intensity of interaction and should enable to play games that conform to playability guidelines.

As the playability of player environments for SGs mainly depends on the quality of played games, in this study we will focus on their usability. Not many authors have evaluated the usability of player environments for SGs. Gaeta et al. (2014) evaluated the usability of a Storytelling Complex Learning Object and found the mean SUS score to be 64.13 which corresponds to a user-friendliness between "ok" and "good" (Bangor, Kortum, & Miller 2008).

Since the EMERGO player environment is software and the quality of the software may influence perceived usability, we will use ISO/IEC 25010:2011 (ISO/IEC, 2011) as the

theoretical framework by which we will explain our findings. Its product quality model is composed of eight software quality characteristics (s1-s8) of which usability (s1) is one. The seven other characteristics are functionality (s2), reliability (s3), performance efficiency (s4), compatibility (s5), security (s6), maintainability (s7) and portability (s8). Perceived usability may be influenced by the player environment's functionality, reliability or performance efficiency, but probably not by its compatibility, security, maintainability and portability because these characteristics are only relevant for its developers but not for its users. ISO/IEC 25010:2011 subdivides usability into six aspects (u1-u6): understandability (u1), learnability (u2), operability (u3), user error protection (u4), user interface aesthetics (u5) and accessibility (u6). (For better readability we have replaced characteristic 'functional suitability' with functionality and 'appropriateness recognizability' with understandability).

3 EMERGO

We developed the web-based EMERGO platform (Nadolski et al., 2008) to support the development and delivery of so called scenario-based SGs where learners are confronted with realistic, ill-defined problems that often allow for multiple solutions and require the application of methodologies or tools and collaboration to get solved (Westera, Nadolski, Hummel, & Wopereis, 2008). The platform offers 22 generic components that support different (didactical) functions that may be needed in scenario-based SGs. It also offers a player environment to play games, an authoring environment to develop games (Slootmaker, Hummel, & Koper, 2017) and environments to monitor students and manage users and game runs. EMERGO has been used to develop 24 games for all kinds of professional and academic fields, and supports the acquisition of four out of five kinds of learning objectives categories as defined by Gagné (1985): intellectual skills, cognitive strategies, verbal information, and attitudes. Motor skills are not supported. The platform is Open source and is available on SourceForge (EMERGO, 2013).

EMERGO games are usually developed by a multidisciplinary team consisting of content matter experts, educational technologists, interaction designers and ICT (Information and Communication Technology) developers. When needed a team is reinforced with other specialists (e.g., for video production). After agreeing on a global description of the game, the team writes the game scenario in three steps, where each step adds more detail. In the end, the scenario describes the tasks to be executed and in what order. Per task it describes which PCs (Playing Characters) and NPCs (Non-Playing Characters) are involved, which resources and tooling are needed, when the task is completed, how this is assessed and which feedback is given in what form and by whom. The authoring environment is used to convert the scenario and materials into game content and script that can be previewed and tested in the player environment.

In the following subsections, we will describe the playing of EMERGO games, the EMERGO player environment and its generic components.

3.1 Playing EMERGO games

In a typical EMERGO game a student enters an authentic environment where he works on professional tasks as a trainee / junior employee. Figures 5.1 and 5.2 give an impression of such a game, in fact one of the two games used in this study. After being welcomed (see Figure 5.1a) the student can navigate to different locations (see Figure 5.1b) to find NPCs like his supervisor, colleagues or experts, or can attend interviews or meetings (see Figure 5.2a). In the environment he has a tablet (see Figure 5.2b) with apps, e.g., a task overview, a resources app, an (in-game) email app or an app to conduct tests. He also has a memo recorder to record interesting parts of interviews and meetings, and a notepad to make contextualized notes. On his tablet he has a memo player app to play his memo recordings and a logbook app to inspect his notes.

The student gets tasks from his supervisor or other NPCs and can send in his outcomes by email, either to NPCs or to PCs (fellow students or teachers). He can be assessed on every action he performs (e.g., which interviews he attends) or by using tests. Depending on actions or progress, the game script may adapt the environment on a micro level (e.g., by (un)locking locations or releasing new resources) or on macro level (e.g., by introducing new tasks). The student may get feedback on his performance by NPCs or in tests. This feedback can incorporate mail attachments or release of resources. If a teacher has a PC role, he can give students feedback within the game; otherwise he can give feedback by impersonating an NPC. The student gets navigation support through alerts (e.g., instructions where to go next).

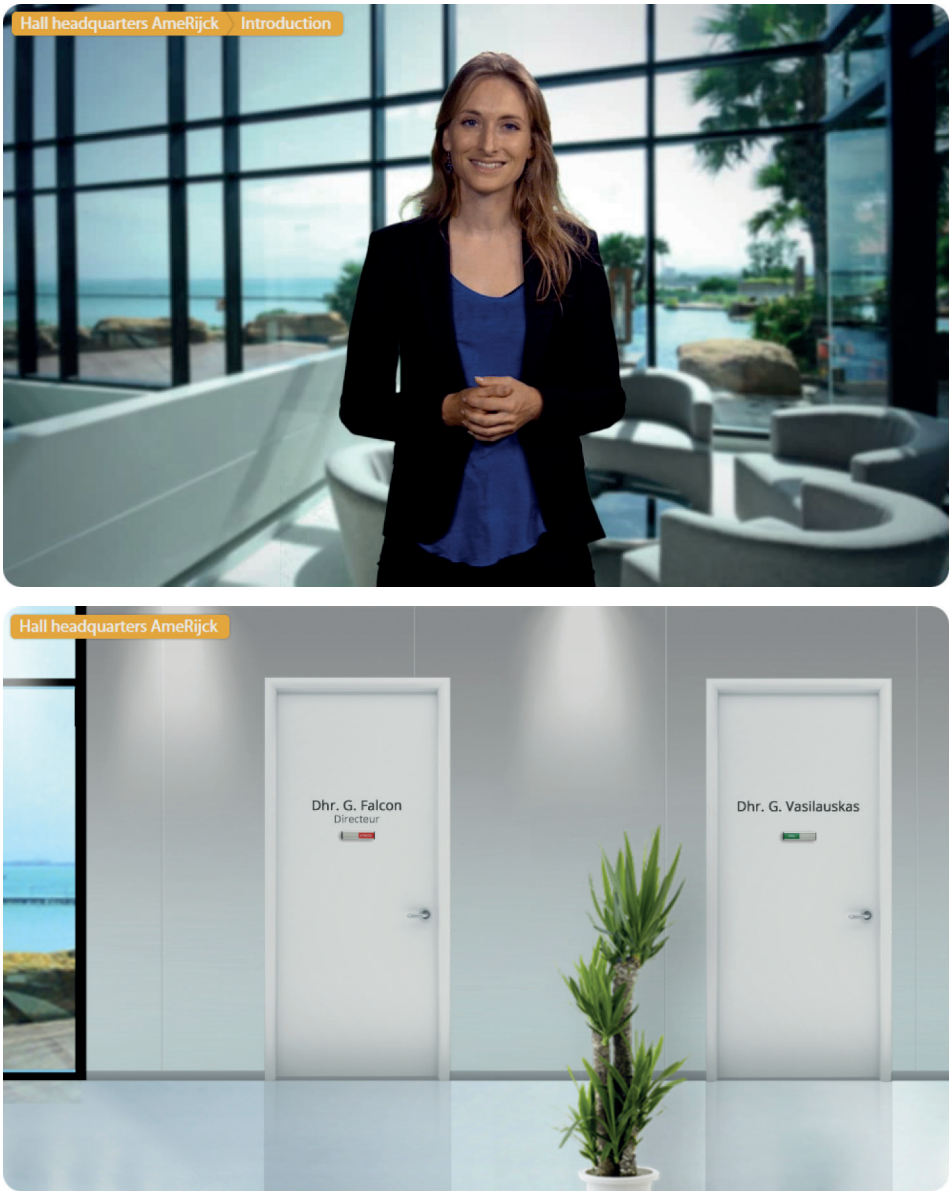

Figure 5.1. Impression of a typical EMERGO game. From top to bottom we see a. an introduction video, and b. a hallway to navigate to different locations

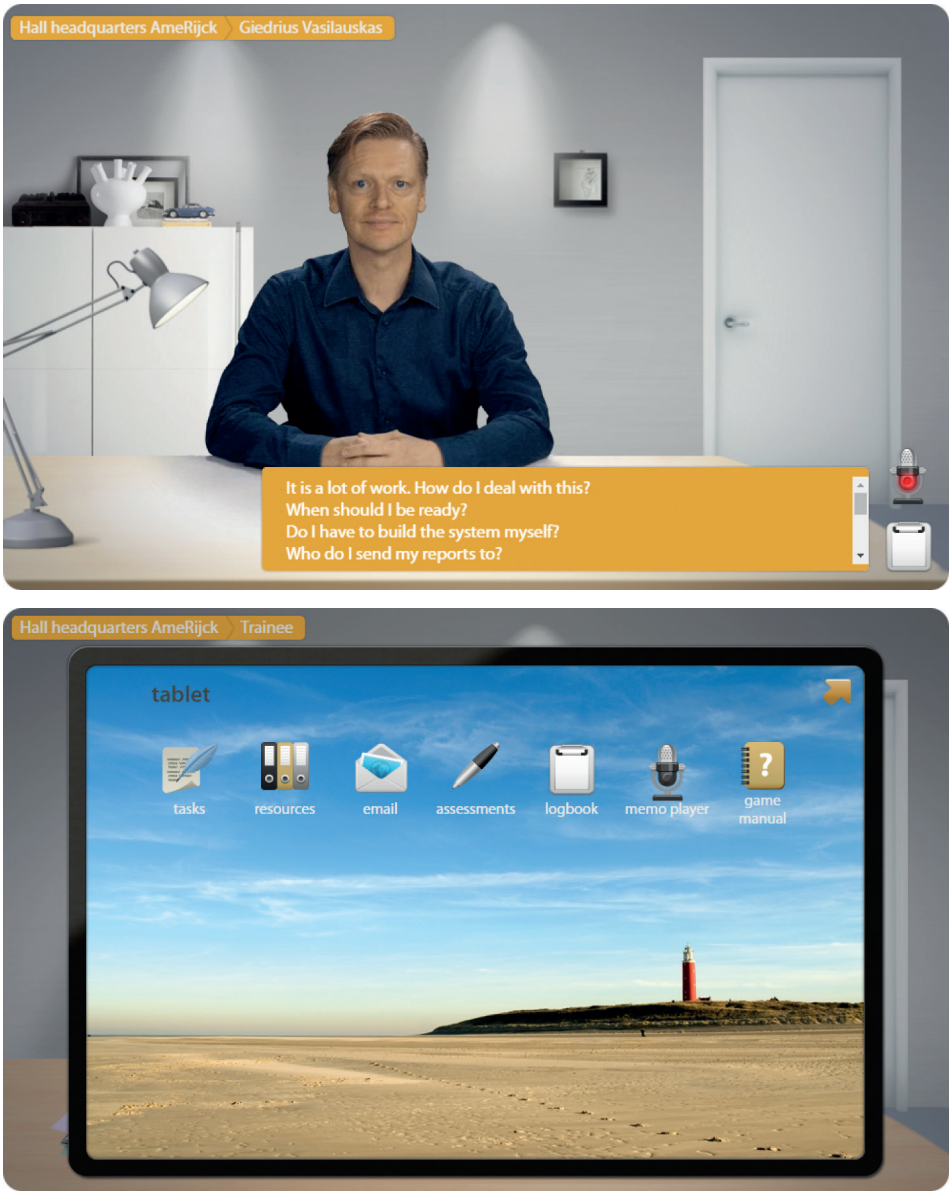

Figure 5.2. Impression of a typical EMERGO game. From top to bottom we see a. an interview with a supervisor using video, and b. a tablet with apps that are needed to during the game

3.2 The EMERGO player environment

One of our main goals was to develop an intuitive, immersive and reliable player environment that will be adapted according to actions and progress of a student. The environment should support multiple game roles, should offer a set of generic, reusable and adaptable components, and should save all student actions, for game script to operate on, and for evaluation and research purposes. The environment should enable ICT developers to rather easily add new components, by applying a generic component template. The current environment also supports the use of skins and plugins. Skins make it possible to offer external parties their own look and feel, and plugins enable ICT developers to add specific functionality only needed once, e.g., for experiments.

The player environment and its components support most heuristics and guidelines compiled by Federoff (2002) and Ibrahim et al. (2012). However, adding user-generated content is only possible in a few components, reversal of actions is not supported, and games are only re-playable with help of an administrator.

Table 5.1: Player environment components and their possible functions

Component	Description	Functions
1. Navigation	Enable spatial navigation through the game	E
2. Conversations	Enable communication with NPCs using video or text	ETKF
3. Notepad	Enable making contextualized notes	EP
4. Memo recorder	Enable recording of conversations	EP
5. Alerts	Provide popup texts	EFN
6. Notifications	Provide (accumulated) embedded texts	EFN
7. Scores	Provide score overview	EF
8. Profile	Enable sharing profile with PCs	EC
9. Chat	Enable communication with PCs	EC
10. Tablet	Enable selecting apps	E
11. Tasks	Provide task (completion) overview. App	ET
12. Resources	Enable consulting resources. App	EKF
13. Email	Enable communication with NPCs and between PCs. App	ETKFC
14. Assessments	Enable conducting tests. App	EAF
15. Logbook	Provide overview of notes. App	EP
16. Memo player	Enable playing back of recordings. App	EP
17. Google maps	Enable inspecting maps with markers. App	EK
18. Directing	Enable analyzing communication between NPCs. App	EP
19. Game manual	Provide help on game interface. App	EN
20. Items	Contains questions to be used by the Assessments component	EAF
21. States	Contains states to be used by the Script or Scores component	A
22. Script	Contains rules to assess the learner and adapt the game on micro and macro level	ETKAFPCN

Table 5.1 shows all available player environment components, their description and which different (didactical) functions they support. The eight functions that may be present in scenario-based SGs are: present and adapt the environment (E); assign tasks

and provide task overview (T); present knowledge (K); assess learner (A); provide feedback (F); support processing of information (P); support collaboration (C); and support navigation (N). One component may serve several functions and one function may involve several components. For instance, the *conversations* component can be used to assign a task or to give feedback. And giving feedback involves the *script* component to determine the correct feedback and the *conversations* component to give it.

Within the player environment the *navigation* component (number 1 in Table 5.1) renders the different locations and background objects, and enables navigation by clickable interface elements (e.g., doors). The *conversations* to *tablet* components (numbers 2 to 10) are presented on top of a location and may be present on all locations or specific ones. The *tablet* is opened on top of *conversations* and is used to present the *tasks* to *game manual* tablet apps (numbers 11 to 19). The *items* to *script* components (numbers 20 to 22) are no recognizable entities in the player environment and are either used by another component or to adapt the environment. Most components allow for having multiple instantiations in the player environment, which enables thematically arranging game content (e.g., one *conversations* component per interviewee). Components may be allocated to specific game roles, which allows for a different environment per game role.

The operation of all components is expected to be rather easy. The component interfaces do not present complex concepts, structures or dynamics (Murray, 2004), so we do not expect to find related problems. However, how the (didactical) functions are translated into usable interfaces may leave room for improvement, which is the motive for our usability evaluation.

4 Method

For the summative evaluation of the usability of the EMERGO player environment we used two games on IT administration (in Dutch) that were developed in 2014 by the Dutch Foundation for Practice-based Learning in the context of the SLEM project (SLEM, 2017). In both games, students have to develop an information system, going through five generally accepted phases for solving IT-problems (including needs analysis, writing a functional design, writing a technical design, making a test plan, and reporting). In the second game the system to be developed is more complex and the learner support provided is less substantive than in the first game. Both games are typical examples of EMERGO games (see section 'Playing EMERGO games'), have quite similar game scenarios, have the same look and feel, offer the same interaction style and use the same 16 out of 22 available player environment components: the *navigation*, *conversations*, *notepad*, *memo recorder*, *alerts*, *tablet*, *tasks*, *resources*, *email*, *assessments*, *logbook*, *memo player*, *game manual*, *items*, *states* and *script* component.

4.1 Participants

One hundred and sixty seven students in IT administration from four Dutch Regional Centers for Secondary Vocational Education participated in this research. These students (two female and 165 male; mean age 19.3 years) were all in their second year of study and individually played either game 1 or game 2. Game 1 was not yet embedded in the curriculum and was tested by 86 students from two participating schools, while game 2 was pilot-tested in regular education by 81 students from all four participating schools. Due to not or not seriously filled-in questionnaires, we could use only data of 120 students for our quantitative data analysis, 56 for game 1 and 64 for game 2. However, for our qualitative data analysis we could use data of all students. The survey was conducted in the fall of 2014 and the first half of 2015.

4.2 Data collection

Students participated in the evaluation while sitting in a classroom (with maximally 30 students) in the presence of researchers, partly authors of this article, and their teacher. We used the same data collection process and instruments (in Dutch) for both games. To get a more complete and detailed picture, we opted for a mixed method of quantitative and qualitative methods to collect both more objective and more subjective usability data. We used a pre-questionnaire just before a game session, individual note taking during the game session, a post-questionnaire just after the game session and a group discussion afterwards. In advance, a researcher gave an oral instruction about the evaluation process. During the game session the researchers and teacher were available in case of questions or problems. The group discussion with students was led by researchers following the same procedure for each class.

For the quantitative part we used questionnaires. The pre-questionnaire included a question to determine student's age and three MC (Multiple Choice) questions (interval, 10-point scale) to determine the level of prior ICT skills, entertainment game skills and SGs skills, in order to find out whether the level of these skills would affect the experienced usability, see Table 5.4. The post-questionnaire included: 10 standard MC questions to determine the SUS mean score, see Figure 5.3; five general usability related MC questions, see Table 5.2; and 11 component specific MC questions, see Table 5.3. All these MC questions used an ordinal 5-point Likert scale ranging from "strongly disagree" to "strongly agree". The post-questionnaire also included a MC question to determine the final grade (interval, 10-point scale) for the operation of the player environment, see Figure 5.4. Afterwards we determined students' play time (mean value 13 hours and 58 minutes) from the logging and received given grades for students' needs analysis reports from the participating teachers (only for game 2).

For the qualitative part we used individual note taking, open questions in the post-questionnaire and a group discussion. Our goal was to collect points of improvement

(tips) and points of satisfaction (tops), regarding both operation of the player environment and provided game content. During the game sessions, students were asked to write down these tips and tops. Based on oral students' comments, present researchers also kept a record of tips and tops. In addition, the post-questionnaire included open questions to collect tips and tops. During the group discussion with students researchers also collected tips and tops.

4.3 Data analysis

For the quantitative analysis we used data of 120 students out of the original 167. We could identify not seriously filled in post-questionnaires because students choose the same answer for all SUS questions while answers are expected to fluctuate because of the questions' alternating positive or negative tone. We could use the data of 56 students that played game 1 (non-response 35%) and 64 students that played game 2 (non-response 21%). Next we calculated the mean SUS scores (Sauro, 2011), and analyzed all data using SPSS (Statistical Package for the Social Sciences).

For the qualitative data analysis we used data of all 167 students, whether they completed the game or not. Of the 637 tips and tops collected during and after the game sessions, 277 tips and tops were unclear or related to game content, which is beyond the scope of this publication. The remaining 198 tips and 162 tops related to operation of the player environment did, although different in wording, partly address the same topics. We identified these similar tips and tops and their frequency and related them to usability aspects or other ISO/IEC software quality characteristics and to either the usability in general or to specific component usability.

5 Results and findings

We now present the quantitative and qualitative findings related to our original evaluation goal which was to evaluate the usability of the EMERGO player environment. Results and findings are presented for both evaluated games together. We end with practical guidelines to improve the usability of player environments for SGs.

5.1 Quantitative results

For game 1 one-way ANOVA analysis demonstrates no significant effect of participating school on the SUS mean scores ($F(1,54) = 0.239$, $p = 0.627$, $\eta_p^2 = 0.004$). However, for game 2 one-way ANOVA analysis demonstrates a significant and large effect of participating school ($F(3,60) = 9.304$, $p = 0.000$, $\eta_p^2 = 0.318$). This appears to be caused by one school where students' ($n = 14$) mean ratings are lowest for all SUS items except one. If we leave out this school, one-way ANOVA analysis demonstrates no significant effect of participating school ($F(2,47) = 0.335$, $p = 0.717$, $\eta_p^2 = 0.014$) for game 2. Analysis of students' qualitative remarks provides possible explanations why students of this one

school rated lower : (i) they complained about a mismatch between game and curriculum, (ii) they knew less what they were up to, (iii) they had less time, and (iv) they complained more about inadequate game preparation and technical problems. For the two schools involved in the evaluation of both game 1 and 2, one way ANOVA analysis demonstrates no significant effect of game on the SUS mean scores ($F(1,81) = 2.496$, $p = 0.118$, $\eta_p^2 = 0.030$), so we present our results for both evaluated games together.

Figure 5.3 shows students' SUS mean score to be 57.85, which is below the mean score of 68.05 for web pages and applications as determined by Bangor, Kortum, and Miller (2008) and corresponds to a user-friendliness between "ok" (52.01) and "good" (72.75). The 10 SUS items have a Cronbach's alpha of 0.76, so demonstrate an acceptable internal reliability (Sauro, 2011). The Shapiro-Wilk test ($W = 0.973$, $p = 0.016$) demonstrates that the SUS mean score is not normally distributed. However, for games 1 and 2 separately it is normally distributed, $W = 0.963$, $p = 0.079$ and $W = 0.967$, $p = 0.088$, respectively.

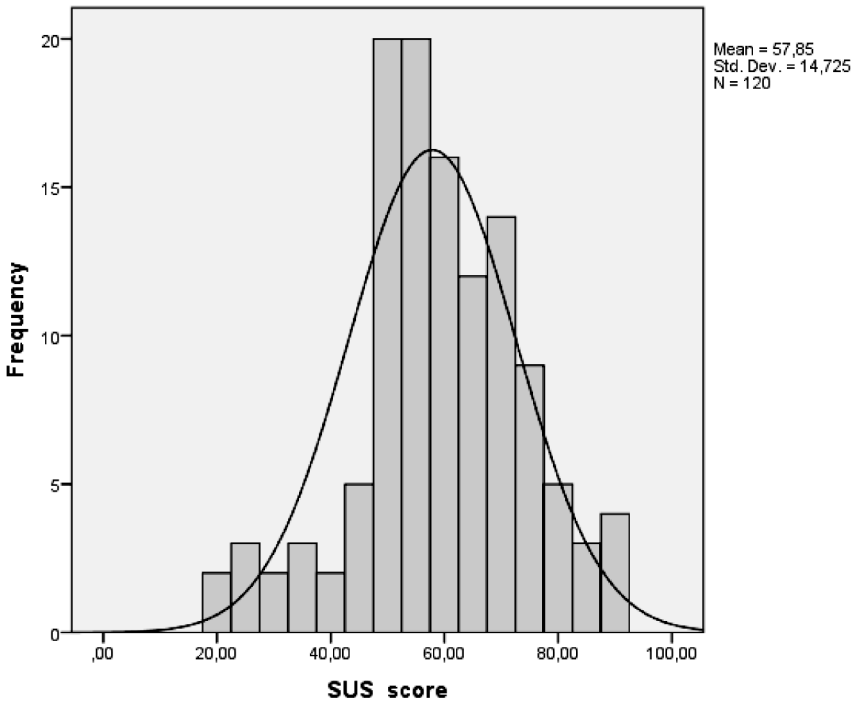

Figure 5.3. Frequency distribution for SUS scores

Table 5.2 shows the results for the five general usability related questions. All mean values are within 3 (“neutral”) and 4 (“agree”). Students seem to be relatively positive about the clearness of the used space metaphor and are relatively negative about the clearness of operating instructions and about having sufficient control.

Table 5.2: General usability questions (n = 120)

Question	*M*	*SD*
1. The space metaphor as a basis for the design of the game is clear	3.99	0.96
2. I find the operating instructions in the game clear	3.47	0.96
3. I could start and shut down the game without any problems	3.77	1.23
4. I think I have sufficient control within the game	3.34	1.07
5. I always know where I am in the game	3.67	0.97
Mean	3.65	0.71

Table 5.3 shows the results for the 11 component specific questions. All mean values are within 3 (“neutral”) and 4 (“agree”). Students seem to be relatively negative about the *memo recorder* component and relatively positive about the *tablet* component. Note that no questions were asked about the *notepad* and *logbook* component.

Table 5.3: Component specific usability questions (n = 120)

Question	Component	*M*	*SD*
1. How I had to navigate within the game was clear	Navigation	3.68	0.99
2. How I had to conduct conversations and interviews within the game was clear	Conversations	3.68	1.11
3. How to record conversations and interviews was clear	Memo recorder	3.28	1.19
4. The operation of popup notifications was clear	Alerts	3.57	1.00
5. How to use the tablet was clear	Tablet	3.86	0.96
6. How to use the task overview was clear	Tasks	3.63	1.09
7. How to use the resources was clear	Resources	3.68	1.02
8. How to use the email was clear	Email	3.82	1.00
9. How I had to use the assessments was clear	Assessments	3.57	1.16
10. How I could play back conversations and interviews was clear	Memo player	3.71	1.09
11. How to use the game manual was clear	Game manual	3.60	1.02
Mean		3.64	0.73

The final grade for operation of the player environment is 6.22, see Figure 5.4. In the Dutch language area a grade of 6 corresponds to "sufficient". The Shapiro-Wilk test (W = 0.927, p = 0.000) demonstrates that the final grades for operation are not normally distributed, also not for games 1 and 2 separately, W = 0.902, p = 0.000 and W = 0.941, p = 0.004, respectively.

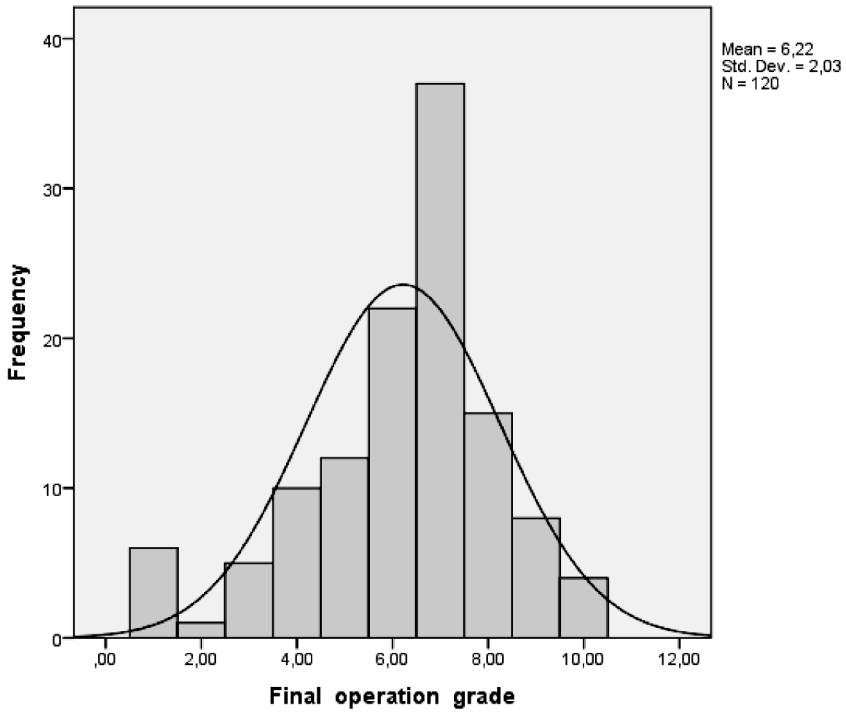

Figure 5.4. Frequency distribution for final grades for operation of the player environment

We found no indication that students' prior skills affected the experienced usability. Table 5.4 shows that Pearson correlation coefficients are not (r <= 0.1, eight cases) or weakly significant (0.1 < r <= 0.3, four cases). We also found no indication that students' age or play time, or grade for needs analysis report (only for game 2) affected the experienced usability. Pearson correlation coefficients are not (r <= 0.1, eight cases) or weakly significant (0.1 < r <= 0.3, four cases).

Table 5.4: Pearson correlation coefficients between students' attributes and usability data

Student attribute	Usability data	Magnitude (*r*)
Prior skills		
ICT skills	Mean SUS scores	0.055
	Mean general usability	0.014
	Mean component specific usability	0.138
	Final grade for game operation	-0.042
Entertainment game skills	Mean SUS scores	-0.015
	Mean general usability	0.062
	Mean component specific usability	0.087
	Final grade for game operation	-0.014
SGs skills	Mean SUS scores	0.111
	Mean general usability	0.139
	Mean component specific usability	0.205
	Final grade for game operation	-0.015
Age	Mean SUS scores	-0.015
	Mean general usability	-0.062
	Mean component specific usability	-0.063
	Final grade for game operation	-0.063
Play time	Mean SUS scores	0.081
	Mean general usability	0.236
	Mean component specific usability	0.181
	Final grade for game operation	0.207
Grade for needs analysis	Mean SUS scores	0.166
	Mean general usability	-0.041
	Mean component specific usability	0.073
	Final grade for game operation	0.057

5.2 Qualitative findings

In general, students have strong opinions and extreme positions (i.e., they can be either very positive or very negative).

Regarding general usability, *understandability (u1)* is somewhat problematic because some students miss operating instructions (11 tips) although others find the game environment to be very clear (four tops). *Operability (u3)* is predominantly valued (80 tops, "Game operation is easy"), but some students miss feedback on download status and errors, or sounds when clicking or in case of alerts (11 tips). *User interface aesthetics (u5)* is somewhat problematic because some students expect a more realistic and dynamic, and less structured environment where they can walk around in 3D instead of clicking on doors and can communicate through talking instead of choosing predefined questions (16 tips), although others find the game environment to be beautifully made, the layout good and the interface nice (nine tops). *Functionality (s2)* is somewhat problematic because some students find the alerts during interviews to be disturbing and miss a timeline and scoring to indicate their progress (five tips). *Reliability (s3)* is predominantly valued (eight tops, "It works fine and as expected"), but some students miss better browser support (three tips). *Performance efficiency (s4)* is problematic (72 tips,

"Slow page loading and game operation, problems while playing video files and progress being not or incorrectly saved"), which was mainly caused by slow wireless internet connections in some schools. We could not relate tips or tops to usability aspects learnability (u2), user error protection (u4) or accessibility (u6), or to other software quality characteristics.

Regarding component usability, the *navigation*'s operability *(u3)* is predominantly valued (15 tops, "It is clear and easy"), although some students found horizontally scrolling to be unhandy (three tips). The *conversations*' operability is predominantly valued (18 tops, "The intuitive way you ask questions is good"), although a number of students miss video controls (15 tips), which, however, are left out deliberately for more realism. Its user error protection (u5) is somewhat problematic, because leaving a location will end a conversation without any warning (nine tips). Its functionality (s2) is also somewhat problematic, because predefined questions may not cover all questions one would like to ask (one tip). The *notepad*'s operability is somewhat problematic, because it cannot be dragged (nine tips). Its functionality is also somewhat problematic, because students see no advantage in making context specific notes (13 tips), although others find it very handy (six tops). The *memo recorder*'s functionality is valued (three tops, "It is very handy"). The *tablet*'s operability is valued (three tops, "It is easy and clear") and its functionality is also valued (four tops, "It works fine"). The *tasks*' functionality is valued (eight tops, "It is important, necessary and handy"). The *resources*' functionality is somewhat problematic, because clicking a link sometimes resulted in two file downloads (nine tips). The *email*'s operability is valued (four tops, "It is easy"). Its reliability (s3) is somewhat problematic, because the input control for mail text was not always rendered in all browsers (seven tips). Its functionality is also somewhat problematic, because feedback on a sent mail is not substantive (two tips). The *assessments*' operability is somewhat problematic, because some students find finishing of an assessment to be cumbersome (six tips). The *logbook*'s functionality is somewhat problematic, because some students find the presentation of notes to be unclear (four tips). The *memo player*'s functionality is also somewhat problematic, because some students miss the original background (two tips). We could not relate tips or tops to the *alerts* and *game manual*, to usability aspects understandability (u1), learnability (u2), user interface aesthetics (u5) and accessibility (u6), or to other software quality characteristics.

5.3 Usability guidelines for player environments for serious games

We present guidelines for player environments for serious games that are based on tips given by students. As our intention was to evaluate the usability of the player environment we had to filter out game usability issues. For instance, students' expectation of a more realistic and dynamic environment where one is more in control is related to both environment (more video game interface alike) and game (more in control). And expe-

rienced inadequate instruction may be related to the environment (operation of the components) or game (navigation and learning support).

Guidelines to improve understandability:

- Match the interface of the environment to the expectations of its intended users as much as possible
- Offer a timeline to show user's progress in time and learning
- Use scoring to encourage the user to perform better
- Let users formulate their own questions using text or speech analysis
- Use text analysis of reports to be able to give substantive feedback
- Present operating instructions at the right time
- Use sounds to support interaction between user and environment
- Always present clear messages in case of errors or time-consuming processes.

Applying the first guideline may not always be possible because a player environment is set up using some kind of metaphor and therefore cannot be expected to serve all kind of games and all kind of users (e.g., young and old), apart from the costs that would be involved.

6 Conclusions and discussion

Our research goal was to evaluate the usability of the EMERGO player environment for scenario-based serious games in detail and to establish guidelines to improve the usability of player environments for SGs. To accomplish this goal, we evaluated the experienced usability of the environment for two games about IT administration. Although both games used only 16 environment components out of 22 two available (73%), we think that our findings are representative for the usability of the player environment as a whole. First, the games are typical examples of EMERGO games, meaning components involved have been used in games most often. Second, the remaining not-used components have comparable interfaces, input controls, and complexities as the used ones.

Students find the general usability of the player environment to be between "ok" and "good". The mean SUS score is 57.85, which is lower than the mean overall score of 68.05 for web pages and applications, (Bangor, Kortum, & Miller, 2008), and the mean given grade for operation of the environment is 6.23, which corresponds to "sufficient". However, in their remarks students are predominantly positive about the environment's operability. Students are relatively negative about its understandability and user interface aesthetics. They miss operating instructions and would like to operate in a more realistic and dynamic and less structured environment where they are more in control. In addition, its functionality is somewhat problematic and its reliability is predominantly valued. Students find the understandability of all components to be "sufficient" although they are relatively negative about the *memo recorder* component and relatively positive about the *tablet* component. Components' operability is predominantly valued, but functionality and reliability are somewhat problematic for some components. We

can identify two possible limitations of our findings. First, our findings do not include all usability aspects and software quality characteristics. However, we think this is not due to our evaluation method but to the fact that students did not mention related remarks, either because they had no problems with it or it was not relevant in their context of use, e.g., they made no remarks on usability aspect 'learnability'. Second, certain qualitative remarks may be overvalued, because students could express same remarks on different occasions.

The found usability is lower than found in earlier superficial evaluations of the player environment (Nadolski et al., 2008; Slootmaker, Kurvers, Hummel, & Koper, 2014). The cause of the lower usability might be that, just as Lewis (2014) stated, usability rather is an emergent property that depends on interactions among users, products, tasks and environments. Because of their age (mean age 19.3 years), students involved probably were more used to playing video games, which may be the reason why they expected a more realistic and dynamic player environment where they are more in control. However, the game scenarios were linear with tasks in a fixed order and missed scoring, levelling or unexpected events. The tasks may have been less challenging or enjoyable, because they had expected outcomes or contained little fun. The games used in the evaluation were used for the first time so possibly had imperfections in playability and were not yet (fully) embedded in regular education, which possibly resulted in inadequate game preparation and students missing instruction during a game session.

We presented usability guidelines for player environments for serious games. Some guidelines, e.g., the use of scoring, correspond with heuristics and guidelines compiled by Federoff (2002) and Ibrahim et al. (2012). Some guidelines seem to be so general that they may also be applicable to other types of player environments.

As a follow-up of this evaluation we plan to re-design the EMERGO player environment according to our findings and to evaluate its usability again in a future study, where we will also involve teachers and admins who are responsible for the embedding of games in education. The results of this evaluation could provide insight on the impact of specific design ideas and implementations on the improvement (or not) of the usability of such a system.

Since the time of study we have extended the player environment with eight new components to, e.g., show user's progress in time and learning and support webcam recordings as a reaction on a shown video, categorization of text or video fragments and forms with different types of input controls. In the near future there might be a need for a mobile or more flexible player environment skin (e.g., in screen, input element and font size) or better accessibility support for people with disabilities.

Acknowledgements

We thank the Dutch Foundation for Practice-based Learning for collaborating in the development and evaluation of the games, and Wim Westera and Henk van den Brink of the Open University of the Netherlands for project management and writing of the game scenarios, respectively. We also would like to thank all students and teachers of the four Dutch Regional Centers for Secondary Vocational Education that participated in this study. This publication has been produced in the context of the SLEM project (SLEM, 2017) which has received funding from the Dutch NRO under grant agreement No. 405-14-504.

Chapter 6

General discussion

1 Introduction

In this thesis we presented the design and evaluation of the EMERGO platform that was developed in the context of online higher education in the Netherlands. The platform aims to enable online universities to efficiently develop and deploy scenario-based serious games for complex cognitive skills acquisition. Serious games are powerful means to provide learning in a more motivating and effective way and their application is still growing. If applied successfully, serious games may increase learners' motivation and may have a positive effect on learning, thereby increasing the effectiveness of education.

The platform provides environments for all stakeholders involved in scenario-based serious games development, namely teachers, students, administrators and ICT developers. Main requirements for the platform were to offer a set of reusable and adaptable components that cover most functionalities needed in scenario-based serious games, to provide a user-friendly authoring environment for teachers and an intuitive and immersive player environment for students. These requirements have been operationalized in the general design question of this thesis:

> *How to design and develop a generic platform for fast and flexible development and delivery of a wide variety of scenario-based serious games that enable complex cognitive skills acquisition?*

Evaluations of teachers using the authoring environment showed that two components were difficult to use and, for one of them, that some teachers had trouble to author game script independently. This made us question how we could improve the usability of the authoring environment and if our findings would be similar to findings for comparable environments. These considerations have been operationalized in the first additional design question:

> *1. How to improve the usability of authoring environments for serious games?*

Evaluations of the player environment showed that students were satisfied to very satisfied about its user interface. However, we did not evaluate the user interface in detail, were curious if students were still satisfied and if our findings would be generalizable to comparable environments. These considerations have been operationalized in the second additional design question:

> *2. How to improve the usability of player environments for serious games?*

In section 2 we summarize our answers to the design questions given in chapters 2 to 5 and draw main conclusions. In section 3 we present recent and future development and research. In section 4 we discuss the significance of the platform.

2 Review of results and conclusions

2.1 The general design question

In chapter 2 and 3 we answered the general design question, namely how to design and develop a generic platform that enables fast and flexible development and delivery of a wide variety of scenario-based serious games that enable complex cognitive skills acquisition.
We described the main design steps that had to be taken before the platform could be implemented, which included: (i) identify the intended users of the platform, (ii) set up requirements for the intended users and the platform in general, (iii) choose an appropriate platform architecture and supporting technologies, (iv) identify needed platform roles, (v) set up a domain model for the platform, (vi) identify an initial set of platform components needed, (vii) set up a generic component design, and (viii) choose a method to implement game script authoring that requires no programming.

Evaluation showed that virtually all of the initial requirements for the platform were met.

We managed to design and develop a generic, fast, flexible, reliable and sustainable platform.

The platform is *generic* because it enables a broad variety of game scenarios to be authored, to be played and to be monitored as is demonstrated by 26 games developed for six content domains with study loads ranging from two to 30 hours. The platform also integrates development and delivery tasks that require different types of platform users in one system and can be used for both education and research purposes.

The platform is *fast* because teachers can use the authoring environment for the most part independently, can draw on already developed components and can preview and test a game in every stage of authoring. The result is more efficient development, as indicated by a decrease in production time by a factor three to four as compared with values found before. In addition, web-based delivery ensures fast and easy delivery of games and of updates of games and the platform itself.

The platform is *flexible* because it allows for games having multiple authors and for parts of game script to be switched on or off during game sessions. In case of bugs authors can adjust already deployed games and teachers can interfere in a running game. In case students have problems the platform allows administrators to fix these problems by changing students' progress data. In addition, developed games and game components can be easily distributed to other platform instances and the platform can be rather easily extended with new components.

The platform's *reliability* and *sustainability* are demonstrated by the fact that it has been used for over 10 years to develop many new games used by thousands of students

in total and that the first games developed are still used in education, and by the increase of the number of platform components from initially 12 to 30 at the moment. Extendibility is facilitated by (i) the generic component design, which serves as a template for defining new platform components and their structure, properties, content and interdependencies, (ii) the flat structure of the components where dependencies are defined by relations rather than by a hierarchy and (iii) the platform's multilayered architecture that neatly separates different responsibilities and processes.

We did not fully manage to design and develop a user-friendly authoring environment. Although teachers could author game content almost independently, some of them had trouble to author game script independently. All teachers found two components, the Script and Conversations components, difficult to use. We showed that all game content, including game script, is authored and validated using a single editor that uses a component's XML definition to render the component's content and input elements to manipulate it.

We managed to design and develop an intuitive immersive player environment as indicated by students being satisfied to very satisfied about its user interface. We showed that game script allows for adapting the environment to the individual student with respect to guidance, presentation, navigation support, feedback, adaptive behaviour of NPCs and task sequencing. Specific Java components are responsible for students' progress management and for handling of events, which include student, timer, script and peer events.

2.2 The first additional design question

In chapter 4 we answered the first additional design question, namely how to improve the usability of authoring environments for serious games.

We conducted an in-depth qualitative study of the authoring environment by applying semi-structured interviews, compared our findings on usability with those found for comparable environments in literature and established guidelines to improve the usability of authoring environments for serious games in general.

We found usability aspects *understandability* and *learnability* of the authoring environment to be problematic, which is in line with findings for comparable environments that also show shortcomings with regard to these aspects, and found its *operability* to be somewhat problematic. We found the environment's functionality and reliability to be valued and its use to be well integrated in the EMERGO method.

Problems with usability are caused by a lack of guidance and support, and by the inherent complexity of the serious games domain and the environment's components. Guidance and support can be improved by adding them. However, inherent complexity, which is determined by the number of abstract concepts, complex structures or dynam-

ic structures that have to be understood, maintained or tested, may be hardly influenced by usability improvements. Based on the work of Murray (2004, 2016) we estimated that most EMERGO components have a low complexity, five have a medium complexity and one, the Script component, has a high complexity.

The amount and form of needed guidance and support will depend on the complexity capacity of the user, e.g., in general teachers have a lower complexity capacity than ICT developers who are used to design and debug structural and procedural models. A related aspect is the abstraction level of the environment as a whole and of its various components. It is an indication of the extent to which authors work with constructs that are more high level, such as tasks or characters, or more low level, such as programming instructions (Dörner, Göbel, Effelsberg, & Wiemeyer, 2016). The abstraction level of the EMERGO authoring environment is quite low (although it presents many abstract concepts). The composition and cooperation of components and entering content are quite low level and entering game script is close to programming. The quite low abstraction level of the environment is both a strength and a weakness. It makes the environment more flexible on the one hand but on the other hand more difficult to comprehend, which may be one of the reasons why teachers found some components difficult to use.

Based on remarks of respondents we established guidelines to improve the usability of authoring environments for serious games with respect to understandability and learnability.

To improve understandability we recommend to: (i) simplify authoring by offering an intuitive user interface, which might be different for different kinds of authors, (ii) reduce complexity by offering two levels of input, basic for novices and advanced for experts, (iii) offer examples of scenarios, games, and game components and how they relate to each other, so authors better understand what to do, (iv) offer a preview option to preview entered content at any time, so authors better understand what they are doing, and (v) use clear terminology fitting authors' expectations.

To improve learnability we recommend to: (i) offer clear instruction and wizards to guide authors during the authoring process, and (ii) offer information on didactics and use of components, so novice authors can make a quick start.

Most guidelines seem obvious but can easily be neglected under time pressure or due to other causes. Further, the guidelines seem to be so general that they may also be applicable to other types of authoring environments.

2.3 The second additional design question

In chapter 5 we answered the second additional design question, namely how to improve the usability of player environments for serious games.

We conducted a mixed method study of the player environment, using quantitative and qualitative methods to collect both more objective and more subjective and detailed usability data, and established guidelines to improve the usability of player environments for serious games in general.

We found the usability of the player environment to be quite low. We found the mean SUS (System Usability Scale) score of the environment to be 57.85, which is lower than the mean score of 68.05 for web pages and applications as determined by Bangor, Kortum, and Miller (2008), and corresponds to a usability between "ok" and "good". The mean given grade for operation of the environment is 6.23, which is rather low and corresponds to "sufficient". However, in their remarks students are predominantly positive about the environment's *operability*. Students are relatively negative about its *understandability* and *user interface aesthetics*. They miss operating instructions and would like to operate in a more realistic, more dynamic and less structured environment where they are more in control. In addition, the environment's functionality is somewhat problematic but its reliability is predominantly valued. Students find the *understandability* of the various platform components "sufficient". The *operability* of the components is predominantly valued, but functionality and reliability of some components is somewhat problematic.

The found low usability may be caused by students' young age (mean age 19.3 years) in comparison to earlier evaluations (see chapter 2), causing them to be more familiar with playing video games, which may be the reason why they expected a more realistic and dynamic player environment where they are more in control. Another cause might be that the games used to evaluate the player environment were too restrictive or not challenging, enjoyable or exciting enough. The games might also have had imperfections or have been introduced inadequately because they were tested with students for the first time. So although the platform and its components make it possible to apply almost all usability heuristics and playability guidelines compiled by Federoff (2002) and Ibrahim et al. (2012) game developers may omit to do so.

Based on remarks of students we established guidelines to improve the usability of player environments for serious games: (i) match the interface of the environment to the expectations of its intended users as much as possible, (ii) offer a timeline to show user's progress in time and learning, (iii) use scoring to encourage the user to perform better, (iv) let users formulate their own questions using text or speech analysis, (v) use text analysis of reports to be able to give substantive feedback, (vi) present operating instructions at the right time, (vii) use sounds to support interaction between user and environment, and (viii) always present clear messages in case of errors or time-consuming processes. Guidelines (iii) and (vii) correspond with heuristics and guidelines compiled by Federoff (2002) and Ibrahim et al. (2012).

2.4 Conclusions

We have demonstrated how to design and develop a generic platform for fast and flexible development and delivery of a wide variety of scenario-based serious games that enable complex cognitive skills acquisition. More cost-efficient development makes it possible to serve smaller target groups, which are often found in higher education, e.g., for a specific course in a specific content domain. Authoring of game script without programming and the possibility to play and test games during development anytime serve specific serious game development team members like educators and domain experts. Adaptation to the individual student, game evaluation and research on learning effects are served by extensive student logging.

We have demonstrated the needs for improving the usability of authoring environments for serious games and established guidelines to improve their understandability and learnability. These guidelines can be used to improve the EMERGO authoring environment that proved to be problematic with respect to understandability and learnability. Despite this lower usability teachers were able to develop serious games and even more efficient than before. However, authoring of game script proved to be too difficult for some teachers and due to its inherent dynamic complexity probably could be better done by a more technically skilled person.

We have demonstrated the needs for improving the usability of playing environments for serious games and established guidelines to improve their usability. These guidelines can be used to improve the EMERGO player environment that proved to have a lower usability than found in previous evaluations. Despite this experienced lower usability most students managed to successfully complete the games used in the evaluation.

3 Recent and future development and research

3.1 Recent development and research

Since the evaluation studies of the authoring and player environment we have improved the platform by adding new platform components, extending the platform's role environments and integrating other tooling. A part of these improvements has been triggered by the increasing demand to use the platform for research purposes.

The platform is extended with eight generally usable platform components.

Seven components are used in a serious game developed for an introductory course in Psychology, where students become acquainted with four fields within Psychology. The game is also used for educational research into various game conditions that vary in activation level and richness of context. Students play 16 mini-games where they visit 12 professionals who work in one of these four fields and who ask them to perform tasks. A so-called dashboard component shows a student's experienced satisfaction and

task difficulty, and monitored performance on eight different skills practiced within the game. The other six components are used to perform tasks that are directly related to the professional's field of work and offer functions that allow for categorization of text or video fragments and filling in forms with different types of input controls.

The so-called iSpot component is used within two courses about conversation skills and allows students to practice these skills by making webcam recordings of their reactions on shown videos, e.g., a patient having a problem. Fellow students or teachers playback and asses these recordings and use the webcam to record their feedback. Students can use these feedback recordings to improve their performance until they make final webcam recordings that are used for examination. The iSpot component has also been used to collect research data for the RAGE project (http://rageproject.eu/).

We improved the environments for a number of platform roles.

For the developer role the Script component now enables to create template scripts where script conditions and actions do not relate to game content but to ids or patterns of ids of game content that does not have to exist yet. The actual game content involved in these template scripts is determined during a game session. This option allows for creating generic game script that can be used in similar situations but with different game content involved, this way decreasing redundancy of script.

The run manager role now has an option to initialize or adjust game content properties within students' progress before or during a run. This option allows for differentiating runs of the same game, e.g., in case of experiments where different runs should meet different experimental conditions, and for changing the player environment for all students at the same time, e.g., to switch to a next level or task.

The tutor role now has an option to generate his own overviews of specific game content properties, e.g., to inspect specific students' performance indicators or paths through the game.

We did some work on integration of the platform with other tooling.

The platform now supports integrating questionnaires made in Google Forms and LimeSurvey that can be used for data collection in research experiments. Game script is used to start a questionnaire at the right moment and to check if it has been completed.

We integrated a RAGE Adaptation and Assessment component to automatically adapt game difficulty to a student's expertise (Van der Vegt, Nyamsuren, Kurvers, & Westera, 2018). By using this component game authors do not have to create game script anymore to support this kind of adaptation.

The platform now uses a separate streaming server (Wowza Streaming Engine) for storage and streaming of all video content including students' webcam recordings.

3.2 Future development and research

Apart from improving the usability of the platform's authoring and player environments based on the presented guidelines, this thesis opens various other possibilities for future development and research. To meet the growing demand for the application of serious games in online education, we now will explore options to improve the authoring and player environment and the platform in general.

To guide game developers through the design and authoring process we could develop a scenario editor component. This component would act as a wizard and would lower the threshold for serious game development by supporting instructional design and authoring by making use of templates, graphical representations and visualisations. Authoring is simplified because this component will function as an abstraction layer above the other rather low level components and can be used to orchestrate them, e.g., by combining adequate learning - with gaming mechanics. Such a scenario editor component would allow for using the authoring environment in a more agile manner and much earlier in the development process, even directly after game ideation so without the necessity of writing a textual scenario beforehand. Adaptation techniques applied for students playing serious games could be used to adapt the scenario editor to the individual game developer. Another promising possibility could be that accumulated student data would be used to improve design advice within the scenario editor.

To improve the usability of game script authoring we could implement a more graphical or block-based interface like in MIT's Scratch. However, it remains a challenge to facilitate keeping a good overview in case of a lot of game script. The authoring environment already allows for distributing game script over multiple Script components but in case of dozens of Script components, such as in a serious game for Psychology, keeping a good overview is cumbersome.

A follow-up study could show whether these improvements indeed lead to a better usability of the authoring environment.

To improve the player experience of the player environment we could add better support for using animations and apply responsive design, which guarantees a good rendering on multiple devices having different screen resolutions. To improve the understandability of the environment we could add an option to let the environment and all of its visible components introduce themselves and their working on first encounter. To better support students in acquiring knowledge we could add an option to get in contact with NPCs, using various communication facilities. Within EMERGO games NPCs are often played by domain experts that are interviewed about their work using predefined questions. This option would enable students to ask other not foreseen questions that possibly could be added in the interview later on. A follow-up study could show whether these improvements indeed lead to a better usability of the player environment.

We also could improve the efficiency of student support in case of problems. The current email traffic regarding substantive, functional, or technical problems could be replaced by a non-intrusive option within the player environment that enables students to report any problem on the exact spot where they experience it. This option could also be used to suggest possible improvements. Using an overview of these problems and improvements helpdesk personnel could jump right away to the correct spot in the player environment to diagnose and fix a problem more quickly than before, and to report the solution to a student on the same spot. Game evaluators could use the same overview to inspect the context of a suggested improvement. This may lead to quicker improvements of the player environment or delivered games.

To better support game evaluation we could add a general option to rate a task or level right after completion, e.g., to measure enjoyment, complexity or needed study time. We already applied this option in the serious game for Psychology but it certainly would be useful for other games.

To better support ICT developers in extending the player environment with new components we could offer interface building blocks based on macros or templates to speed up the construction of a new component.

To improve the EMERGO platform in general we are considering adding an extra platform role 'researcher' with its own working environment that can be used for educational research and game evaluation. In recent years the platform has been increasingly used for research where researchers use an existing game or develop their own small dedicated game for data collection. In both cases they require their own specific overviews of combined student data. The current procedure is to assign the platform role 'tutor' to these researchers and to build a specific landing page for them showing the desired student data. We have to investigate if the required functionality in these specific landing pages can be generalized to one working environment. We might integrate the RAGE Gaming Analytics Suite component that is meant to gather and store specific interaction data for real-time performance and progress monitoring, and evaluation. In addition, researchers should also be able to harvest data arbitrarily from all available logging data, which would require integrating an existing data mining tool that offers data mining techniques such as cluster analysis, pattern recognition or outlier detection. Our experience during the development of the serious game for Psychology is that researchers may also need high level (performance) indicators. These indicators require some game specific pre-processing of student progress, e.g., certain competency levels that are determined by a combination of student data that may have their own weighting. For this purpose, we are considering adding a platform component to construct, handle and store these game specific high level indicators.

We could improve integration of the platform with other game and e-learning platforms, external web services and sensors.

We could explore integration with Unity games. We have already experimented with integrating the Unity Web Player where a Unity demo game was played embedded within the platform and events sent to each other triggered each other's adaptation. It would be interesting to further explore this possibility, where the platform could function as an educational shell around a Unity game.

We could better support integration with institutional e-learning platforms. It should be possible to retrieve students' game progress data, e.g., scores, task completion rates or times and task success rates, so this data can be shown in tutor overviews within the e-learning platform.

We could explore integration with external web services for enriching the platform with real-time data and sensors for better student support. We have already built web services that support exchange of student data and that were used in an experiment to trigger specific game support based on a student's mood determined by webcam analysis. It would be interesting to explore integrating other types of sensors and possibly social network activity.

4 Significance of the platform

The platform allows for developing serious games in which the learner is in control, learning is situated, authentic and may be based on a didactical model, and transfer of learned skills to practice is supported. Learners' skills may be improved by offering challenging, exciting and customized tasks, offering multiple perspectives, giving appropriate immediate feedback, integrating assessment of learning and adapting the game to the individual learner. These aspects can foster motivation and active engagement, and this way can support the intended goals of a serious game. For higher education these goals are increasingly related to the acquisition of complex professional and academic skills. The platform enables to achieve these goals and thus to improve the effectiveness of education.

Due to its technical platform independence and use of Open source frameworks online universities can easily use the platform. Institutes may install their own platform instance or various institutes may use one platform instance where multiple administrators manage their own development and deployment environment. In both cases developers may exchange each other's games and game components by exporting and importing them.

Although the platform and its underlying architecture do not pose any obstacles to wider use it is not widely used. Over the years the platform has been used for prolonged periods of time by three other Dutch institutes for online education. Various other (inter)national institutes for education have come to know the platform of which some have used it for educational material development. Causes of this limited use

might be the difficulty in organizing cooperation between institutes, the ‘not invented here’ syndrome, insufficient promotion or the still high costs of game development.

Future improvements in usability and possible new extensions like a scenario editor to better support the design and authoring process, and a ‘researcher’ role to better support educational research and game evaluation, can promote wider use of the platform and wider application of serious games. Other improvements may be realized by integrating RAGE components that offer not yet supported functions. We might also upgrade certain unique EMERGO components to RAGE components to improve awareness of the platform.

The platform and its underlying ideas, architecture, evaluations and future development as presented in this thesis, can be expected to contribute to new development and research in the fields of serious games development, instructional design and online education.

References

Abt, C. C. (1970). *Serious games*. New York, NY: Viking Press.

Aldrich, C. (2005). *Learning by doing: the essential guide to simulations, computer games, and pedagogy e-learning and other educational experiences*. San Francisco, CA: John Wiley & Sons.

Alessi, S. M., & Trollip, S. R. (2001). *Multimedia for learning: Methods and development*. Needham, MA: Allyn & Bacon.

Apache Tomcat (2017). *Apache Tomcat*. Retrieved November 14, 2017, from http://tomcat.apache.org/

Arnab S., Berta R., Earp J., De Freitas S., Popescu M., Romero M., Stanescu I., & Usart M. (2012). Framing the adoption of serious games in formal education. *Electronic Journal of e-Learning, 10*(2), 159-171.

Backlund, P., & Hendrix, M. (2013). Educational games - Are they worth the effort? A literature survey of the effectiveness of serious games. *Proceedings of the 5th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, 1-8. doi : 10.1109/VS-GAMES.2013.6624226

Bahreini, K., Nadolski, R. J., Qi, W., & Westera, W. (2012). FILTWAM - a Framework for Online Game-Based Communication Skills Training - Using Webcams and Microphones for Enhancing Learner Support. *Proceedings of the 6th European Conference on Games Based Learning*, 39-48.

Bangor, A., Kortum, P. T., & Miller, J. T. (2008). An Empirical Evaluation of the System Usability Scale. *International Journal of Human–Computer Interaction, 24*(6), 574-594. doi:10.1080/10447310802205776

Boyle, E. A., Hainey, T., Connolly, T. M., Gray, G., Earp, J., Ott, M., & Pereira, J. (2016). An update to the systematic literature review of empirical evidence of the impacts and outcomes of computer games and serious games. *Computers & Education, 94*, 178-192. doi:10.1016/j.compedu.2015.11.003

Brooke, J. (1996). System usability scale (SUS). *Usability Evaluation in Industry*. London, UK: Taylor and Francis.

Brooke, J. (2013). SUS: A Retrospective. *Journal of Usability Studies, 8*(2), 29-40.

Bryman, A. (2012). *Social Research Methods, Fourth Edition*. Oxford University Press.

Chin, J. P., Diehl, V. A., & Norman, K. L. (1988). Development of an instrument measuring user satisfaction of the human-computer interface. *Proceedings of the SIGCHI conference on Human factors in computing systems*, 213-218. Washington, DC: ACM. doi:10.1145/57167.57203

Clark, R. C., & Mayer, R.E. (2011). *e-Learning and the Science of Instruction*. New York, NY: Wiley-Blackwell. doi:10.1002/9781118255971

Connolly, T. M., Boyle, E. A., MacArthur, E., Hainey, T., & Boyle, J. M. (2012). A systematic literature review of empirical evidence on computer games and serious games. *Computers & Education, 59*(2), 661-686. doi:10.1016/j.compedu.2012.03.004

Crookall, D. (2010). Serious Games, Debriefing, and Simulation/Gaming as a Discipline. *Simulation & Gaming, 41*(6), 898-920. doi:10.1177/1046878110390784

Cross, N. (2007). From a Design Science to a Design Discipline: Understanding Designerly Ways of Knowing and Thinking. *Design Research Now*, 41-54. doi:10.1007/978-3-7643-8472-2_3

Dede, C. (2009). Immersive Interfaces for Engagement and Learning. *Science, 323*(5910), 66-69. doi:10.1126/science.1167311

De Freitas, S., I. (2006). Using games and simulations for supporting learning. *Learning, media and technology, 31*(4), 343-358.

De Freitas, S., Rebolledo-Mendez, G., Liarokapis, F., Magoulas, G., & Poulovassilis, A. (2010). Learning as immersive experiences: Using the four-dimensional framework for designing and evaluating immersive learning experiences in a virtual world. *British Journal of Educational Technology, 41*(1), 69-85. doi:10.1111/j.1467-8535.2009.01024.x

Djaouti, D., Alvarez, J., Jessel, J. P., & Rampnoux, O. (2011). Origins of Serious Games. In *Serious Games and Edutainment Applications* (pp. 25-43). London, UK: Springer. doi:10.1007/978-1-4471-2161-9_3

Dörner, R., Göbel, S., Effelsberg, W. & Wiemeyer, J. (2016). *Serious Games: Foundations, Concepts and Practice*. Cham, Switzerland: Springer.

Dubey, S. K., & Rana, A. (2010). Analytical Roadmap to Usability Definitions and Decompositions. *International Journal of Engineering Science and Technology, 2*(9), 4723-4729.

Dumas, J. S. (2003). User-based evaluations. In J. A. Jacko & A. Sears (Eds.), *The human computer interaction handbook* (pp. 1093-1117). Mahwah, NJ: Erlbaum.

EMERGO (2013). *EMERGO project page*. Retrieved July 29, 2013 from http://sourceforge.net/projects/emergo/

Federoff, M. A. (2002). *Heuristics and usability guidelines for the creation and evaluation of fun in video games* (Unpublished master's thesis). Indiana, IN: Indiana University.

Fowler, M. (2004). *UML Distilled: a brief guide to the standard object modeling language* (Third Edition). Reading, MA: Addison-Wesley Professional.

Franzwa, C., Tang, Y., Johnson, A., & Bielefeldt, T. (2014). Balancing Fun and Learning in a Serious Game Design. *International Journal of Game-Based Learning, 4*(4), 37-57. doi:10.4018/ijgbl.2014100103

Gaeta, M., Loia, V., Mangione, G. R., Orciuoli, F., Ritrovato, P., & Salerno, S. (2014). A methodology and an authoring tool for creating Complex Learning Objects to support interactive storytelling. *Computers in Human Behavior, 31*, 620-637. doi:10.1016/j.chb.2013.07.011

Gagné, R. (1985). *The Conditions of Learning and Theory of Instruction*. New York, NY: Holt, Rinehard, and Winston.

Games Monitor (2015). *Games Monitor*. Retrieved February 24, 2018, from https://www.dutchgamegarden.nl/project/games-monitor/

Gerrichhauzen, J. T. G., Hoefakker, R. E., Perreijn, A. C., Van den Brink, H. J., Slootmaker, A., & Berkhout, J. (1998). *Buiten dienst* [Out of order] (version 1.0) [multimedia CD-ROM]. Heerlen, The Netherlands: Open University of the Netherlands.

Göbel, S., Salvatore, L., Konrad, R., & Mehm, F. (2008). StoryTec: A Digital Storytelling Platform for the Authoring and Experiencing of Interactive and Non-linear Stories. In Spierling, U., Cavazza, M., Peinado, F., Aylett, R., Swartjes, I., Kudenko, D., Young, R., Tychsen, A., Pizzi, D., El-Nasr, M. (eds.) *Interactive Storytelling*. LNCS, vol. 5334 (pp. 325-328). Berlin – Heidelberg, Germany: Springer. doi:10.1007/978-3-540-89454-4_40

Hamari, J., Shernoff, D. J., Rowe, E., Coller, B., Asbell-Clarke, J., & Edwards, T. (2016). Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning. *Computers in Human Behavior, 54*, 170-179. doi:10.1016/j.chb.2015.07.045

Hartson, R., & Pyla, S. (2012). *The UX Book: Process and Guidelines for Ensuring a Quality User Experience*. New York, NY: Elsevier. doi:10.1145/2347696.2347722

Herrington, J., Reeves, T.C., & Oliver, R. (2007). Immersive learning technologies: Realism and online authentic learning. *Journal of Computing in Higher Education, 19*(1), 80-99. doi:10.1007/BF03033421

Hertzum, M., Hansen, K. D., & Andersen, H. H. K. (2009). Scrutinising usability evaluation: Does thinking aloud affect behaviour and mental workload? *Behaviour & Information Technology, 28*, 165-181.

Hibernate (2017). *Hibernate ORM*. Retrieved November 14, 2017, from http://hibernate.org/orm/

Hommes, M. A., Houtmans, M. A., Hummel, H. G. K., Kuntze, A. J. , Tiesnitsch, D., Rickhoff, M. L., Westera, W., Kerstjens, W. M. J., Kurvers, H. J., Slot, W. J. J., Vander Meeren, W. M. F., & Berkhout, J. (2000). *Diagnosticus* [Diagnostician] (version 1.0) [multimedia CD-ROM]. Heerlen, The Netherlands: Open University of the Netherlands.

Hornbæk, K. (2006). Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Human-Computer Studies, 64*, 79-102.

Hummel, H. G. K., Van Houcke, J., Nadolski, R. J., Van der Hiele, T., Kurvers, H. J., & Löhr, A. (2010). Scripted collaboration in serious gaming for complex learning: Effects of multiple perspectives when acquiring water management skills. *British Journal of Educational Technology, 42*(6), 1029-1041. doi:10.1111/j.1467-8535.2010.01122.x

Hummel, H. G. K., Geerts, W., Slootmaker, A., Kuipers, D., & Westera, W. (2013). Collaboration scripts for mastership skills: Online game about classroom dilemmas in teacher education. *Interactive Learning Environments, 23*(6), 670-682. doi:10.1080/10494820.2013.789063

Huysse, P., Nadolski, R. J., Oldenboom, E., Kerstjens, W. M. J., De Vries, F. J. J, Slootmaker, A, Jordense, M., Jambos, M., & Berkhout, J. (1998). *Paradise Parks* (version 1.0) [multimedia CD-ROM]. Heerlen, The Netherlands: Open University of the Netherlands.

Ibrahim, A., Vela, F. L. G., Rodríguez, P. P., Sánchez, J. L. G., & Zea, N. P. (2012). Playability Guidelines for Educational Video Games: A Comprehensive and Integrated Literature Review. *International Journal of Game-Based Learning (IJGBL), 2*(4), doi:18-40. 10.4018/ijgbl.2012100102

IMSCP-IM (2007). *IMS Content Packaging Information Model*. Retrieved March 01, 2007, from http://www.imsglobal.org/content/packaging/cpv1p1p2/imscp_infov1p1p2.html

ISO (2006). *ISO 9241-110:2006*. Ergonomics of human-system interaction – Part 110: Dialogue principles.

ISO/IEC (2011). *ISO/IEC 25010:2011*. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models.

Ivens, W. P. M. F., Lansu, A. L. E., Hummel, H. G. K., Huisman, W. H. T., Westera, W., Wagemans, L. J. J. M., Slootmaker, A., & Berkhout, J. (1998). *Bodem en milieu* [Soil and environment] (version 1.95) [multimedia CD-ROM]. Heerlen, The Netherlands: Open University of the Netherlands.

J2EE (2017). *Java Platform, Enterprise Edition*. Retrieved November 14, 2017, from http://www.oracle.com/technetwork/java/javaee/overview/

Kickmeier-Rust, M. D., & Albert, D. (2010). Micro-adaptivity: Protecting immersion in didactically adaptive digital educational games. *Journal of Computer Assisted Learning, 26*(2), 95-105. doi:10.1111/j.1365-2729.2009.00332.x

Kickmeier-Rust, M. D., & Albert, D. (2012). Educationally adaptive: Balancing serious games. *International Journal of Computer Science in Sport, 11*(1), 1-10.

Kickmeier-Rust, M. D., Mattheiss, E., Steiner, C., & Albert, D. (2011). A Psycho-Pedagogical Framework for Multi-Adaptive Educational Games. *International Journal of Game-Based Learning, 1*(1), 45-58. doi:10.4018/ijgbl.2011010104

Kirakowski, J. (1996). The Software Usability Measurement Inventory: Background and usage. In P. Jordan, B. Thomas, & B. Weerdmeester (Eds.), *Usability evaluation in industry* (pp. 169-178). London, UK: Taylor & Francis.

Klemke, R., Van Rosmalen, P., Ternier, S., & Westera, W. (2015). Keep it simple: Lowering the barrier for authoring serious games. *Simulation & Gaming, 46*(1), 40-67. doi:10.1177/1046878115591249

Kultima, A. (2015). Game design research. *Proceedings of the 19th International Academic Mindtrek Conference*, 18-25. doi:10.1145/2818187.2818300

Kultima, A., & Sandovar, A. (2016). Game design values. *Proceedings of the 20th International Academic Mindtrek Conference*, 350-357. doi:10.1145/2994310.2994362

Leinders, J. J. M., Drury, S. A., Rothery, D. A., Nadolski, R. J., Van der Heijden, M. P., De Vries, F. J. J., Slot, W. J. J., Roosendaal, A., Kurvers, H. J., & Berkhout, J. (1993). *Reseat* (version 1.0) [multimedia CD-ROM]. Heerlen, The Netherlands: Open University of the Netherlands.

Lewis, J. R. (1995). IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction, 7*(1), 57-78. doi:10.1080/10447319509526110

Lewis, J. R. (2002). Psychometric Evaluation of the PSSUQ Using Data from Five Years of Usability Studies. *International Journal of Human–Computer Interaction, 14*(3), 463-488. doi:10.1207/S15327590IJHC143&4_11

Lewis, J. R. (2014). Usability: Lessons Learned… and Yet to Be Learned. *International Journal of Human-Computer Interaction, 30*(9), 663-684. doi:10.1080/10447318.2014.930311

Marchiori, E. J., Torrente, J., del Blanco, Á., Moreno-Ger, P., Sancho, P., & Fernández-Manjón, B. (2012). A narrative metaphor to facilitate educational game authoring. *Computers & Education, 58*(1), 590-599. doi:10.1016/j.compedu.2011.09.017

Martens, R., Gulikers, J., & Bastiaens, T. (2004). The impact of intrinsic motivation on e-learning in authentic computer tasks. *Journal of Computer Assisted Learning, 20*(5), 368-376. doi:10.1111/j.1365-2729.2004.00096.x

Mehm, F., Göbel, S., Steinmetz, R. (2012). Authoring of Serious Adventure Games in StoryTec. In: Göbel, S., Müller, W., Urban, B., Wiemeyer, J. (eds). E-Learning and Games for Training, Education, Health and Sports. *Lecture Notes in Computer Science 7516*, 144-154. Berlin – Heidelberg, Germany: Springer. doi:10.1007/978-3-642-33466-5_16

Michael, D. & Chen, S. (2006). *Serious Games: Games that Educate, Train, and Inform*. Boston, MA: Thomson.

Microsoft (2009). *Microsoft Application Architecture Guide*. Second Edition. Microsoft Corporation.

Murray, T. (1999). Authoring intelligent tutoring systems: An analysis of the state of the art. *International Journal of Artificial Intelligence in Education, 10*, 98-129.

Murray, T. (2004). Design tradeoffs in usability and power for advanced educational software authoring tools. *Educational Technology, 44*(5), 10-16.

Murray, T. (2016). Coordinating the Complexity of Tools, Tasks, and Users: On Theory-based Approaches to Authoring Tool Usability. *International Journal of Artificial Intelligence in Education, 26*(1), 37-71. doi:10.1007/s40593-015-0076-6

MySQL (2017). *MySQL Enterprise Edition*. Retrieved November 14, 2017, from https://www.mysql.com/products/enterprise/

Nadolski, R. J., Hummel, H. G. K., Van den Brink, H. J., Hoefakker, R. E., Slootmaker, A., Kurvers, H. J., & Storm, J. (2008). EMERGO: A methodology and toolkit for developing serious games in higher education. *Simulation & Gaming, 39*(3), 338-352. doi:10.1177/1046878108319278

Nadolski, R. J., Hummel, H. G. K., Slootmaker, A., & Van der Vegt, W. (2012). Architectures for Developing Multiuser, Immersive Learning Scenarios. *Simulation & Gaming, 43*(6), 825-852. doi:10.1177/1046878112443323

Nielsen, J. (1994). *Usability Engineering*. New York, NY. Elsevier.

Object Management Group. (2017). *Unified Modeling Language (UML)*. Retrieved September 28, 2017, from http://www.uml.org/

Oja, M. K. (2010). Designing for collaboration: Improving usability of complex software systems. *Proceedings of the 28th International Conference Extended Abstracts on Human Factors in Computing Systems*, 152-158. doi:10.1145/1753846.1754059

Olsen, T., Procci, K., & Bowers, C. (2011). Serious Games Usability Testing: How to Ensure Proper Usability, Playability, and Effectiveness. *Proceedings of the International Conference of Design, User Experience, and Usability*, 625-634. Berlin – Heidelberg, Germany: Springer. doi:10.1007/978-3-642-21708-1_70

Paakkanen, V. (2014). *User Experience of Game Development Environments: Can Making Games be as Fun as Playing Them?* Master Thesis. Aalto University, Espoo, Finland.

Pattrasitidecha, A. (2014). *Comparison and Evaluation of 3D Mobile Game Engines*. Master Thesis. Chalmers University of Technology, University of Gothenburg, Göteborg, Sweden.

PMC (2017). *Perspectives from the Global Entertainment and Media Outlook 2017–2021*. Retrieved February 24, 2018, from https://www.pwc.com/outlook

Prensky, M. (2002). The motivation of gameplay: The real twenty-first century learning revolution. *On the horizon, 10*(1), 5-11. doi:10.1108/10748120210431349

Prensky, M. (2007). *Digital game-based learning*. St Paul, MN: Paragon House.

Ryan R. M., & Deci E. L. (2000). Self-determination theory and the facilitation of intrinsic motivation, social development, and well-being. *American Psychologist, 55*(1), 68-78. doi:10.1037/0003-066X.55.1.68

Sauro, J. (2011). *A practical guide to the System Usability Scale: Background, benchmarks, & best practices*. Denver, CO: Measuring Usability LLC.

Schell, J. (2008). *The Art of Game Design: A Book of Lenses*. Second Edition. New York, NY: CRC Press. doi:10.1201/b17723

SLEM (2017). *SLEM project page*. Retrieved July 29, 2017, from https://www.nro.nl/kb/405-14-504-toepassen-serious-games-in-mbo-opleiding-ict-beheerder/

Slootmaker, A., Hummel, H. G. K., & Koper, E. J. R. (2017). Evaluating the usability of authoring environments for serious games. *Simulation & Gaming, 48*(4), 553-578. doi:10.1177/1046878117705249

Slootmaker, A., Kurvers, H. J., Hummel, H. G. K., & Koper, E. J. R. (2014). Developing scenario-based serious games for complex cognitive skills acquisition: Design, development and evaluation of the EMERGO platform. *Journal of Universal Computer Science, 20*(4), 561-582.

Spring Framework (2013). *Spring Framework*. Retrieved November 14, 2013, from http://www.springsource.org/

Tattersall, C., Vogten, H., Brouns, F., Koper, R., Van Rosmalen, P., Sloep, P., et al. (2005). How to create flexible runtime delivery of distance learning courses. *Educational Technology & Society, 8*(3), 226-236.

Theodosiou, S., & Karasavvidis, I. (2015). Serious games design: A mapping of the problems novice game designers experience in designing games. *Journal of e-Learning and Knowledge Society, 11*(3), 133-148.

Thillainathan, N., & Leimeister, J. M. (2014). Serious Game Development for Educators - A Serious Game Logic and Structure Modeling Language. *EDULEARN14 Proceedings*, 1196-1206.

Usability-First (2017). *Playability definition*. Retrieved from: http://www.usabilityfirst.com/glossary/playability

Vandermeeren, W. M. F., Hoogveld, A. W. M., Hummel, H. G. K., Vos, M. M. H. L. S, Rosendaal, A., Van der Vegt, G. W., & Berkhout, J. (1997). *Practicum Assesment Center* (version 3.0) [multimedia CD-ROM]. Heerlen, The Netherlands: Open University of the Netherlands.

Van der Vegt, W., Nyamsuren, E., Kurvers, H. J., & Westera, W (2018). Portable Software Assets for Serious Games: enabling software reuse across programming languages and game engines. Manuscript submitted for publication.

Van Deursen, A., Klint, P., & Visser, J. (2000). Domain-specific languages: an annotated bibliography. *ACM SIGPLAN Notices, 35*(6), 26-36. New York, NY: ACM. doi:10.1145/352029.352035

Van Est, C., Poelman, R., & Bidarra, R. (2011). High-level scenario editing for serious games. *Proceedings of the International Conference on Computer Graphics Theory and Applications*, 339-346. doi:10.5220/0003374503390346

W3C (2015). *Extensible Markup Language (XML)*. Retrieved May 12, 2015, from https://www.w3.org/XML/

W3C (2016). *XML Schema*. Retrieved August 18, 2016, from http://www.w3.org/XML/Schema

W3C (2017). *Simple Object Access Protocol (SOAP)*. Retrieved November 05, 2017, from https://www.w3.org/TR/soap/

Westera, W. (2001). Competences in education: A confusion of tongues. *Journal of Curriculum Studies, 33*(1), 75-88. doi:10.1080/00220270120625

Westera, W., Nadolski, R. J., Hummel, H. G. K., & Wopereis, I. G. J. H. (2008). Serious games for higher education: a framework for reducing design complexity. *Journal of Computer Assisted Learning, 24*(5), 420-432. doi:10.1111/j.1365-2729.2008.00279.x

Westera, W., Nadolski, R. J., & Hummel, H. G. K. (2014). Serious Gaming Analytics: What Students´ Log Files Tell Us about Gaming and Learning. *International Journal of Serious Games, 1*(2), 35-50. doi:10.17083/ijsg.v1i2.9

Westera, W., Slootmaker, A., & Kurvers, H. J. (2014). The Playground Game: Inquiry-based Learning About Research Methods and Statistics. *Proceedings of the 8th European Conference on Games Based Learning, 2*, 620-627.

Wöretshofer, J., Nadolski, R. J., Starren-Weijenberg, A. M. A. G., Quanjel-Schreurs, R. A. M, Aretz, C. C. W. M., van der Meer, N. H. W., Martyn, G., van den Brink, H. J., Slootmaker, A., & Berkhout, J. (2000). *Pleit voorbereid* [Preparing a plea] (version 1.0) [multimedia CD-ROM]. Heerlen, The Netherlands: CIHO.

ZK Framework (2013). *ZK Framework*. Retrieved November 14, 2013, from http://www.zkoss.org/

Appendices

Appendix 1: Estimated contribution of the author to the main tasks presented in this thesis

Chapter	Main task	%
2	Setup of platform requirements	60
	Design of the platform	80
	Implementation of the platform	80
	Evaluation of the platform	20
3	Design and implementation of platform components	80
	Design and implementation of authoring process	80
	Design and implementation of playing process	80
	Design and implementation of other platform process	70
	Setup of platform architecture	70
4	Usability data collection for authoring environment	100
	Usability data analysis for authoring environment	100
5	Usability data collection for player environment	60
	Usability data analysis for player environment	100

Appendix 2: Interview guide

General impression

- What is your general impression of the authoring environment?
- Is it easy to use?
- Is it clear enough?
- Is its subdivision in screens straightforward?
- Is the navigation through screens straightforward?

Requirements (F=Functional, N=Non-functional)

Create and edit games (F)

- Did you create games yourself?
- If so, is editing of games straightforward?
- Did you encounter any problems editing games?
- Did you have any trouble with the concept of games?

Create and edit game roles (F)

- Did you add game roles?
- If so, did you add PCs or NPCs or both?
- Was editing of game roles straightforward, or did you encounter any problems?
- Did you have any trouble with the concept of game roles?

Select and edit game components (F)

- Did you have a good overview of components available beforehand?
- If not, was it difficult to get this overview?
- Did you have any trouble choosing the right components for your game, starting from the scenario?
- Was it obvious to you in which order components should be filled with content?
- Did you miss some functionality or components?
- Could you map your scenario easily to the available components?
- If not, were you able to implement the missing functionality using other components?
- Did you have any trouble with the game component editor in general?
- Did you have any trouble with the concept of the game component?

Developing a game together (F)

- What are your experiences with working together on the same game content?
- Could this process be improved?

Previewing a game or game component (F)

- Did you use the preview option?
- If so, did you use it to preview the game or just a single game component?
- For which game components did you use it?
- Did you have any difficulty using this option?
- Is it straightforward?
- Could it be improved?

Testing a game (F)

- Did you use the preview option for testing?
- If so, did you use it to test the game in total or just a single game component?
- For which game components did you use it?
- Did you use the option to create multiple preview items corresponding to multiple starting points within the game?
- Did you have any difficulty using this option?
- Is it straightforward?
- Could it be improved?

Import or export a game (F)

- Did you copy, import, or export games?
- What are your experiences with it?
- Did you have trouble with it?
- Could it be improved?

Import or export a game component (F)

- Did you copy, import, or export game components?
- If so, which game components?
- What are your experiences with it?
- Did you have trouble with it?
- Could it be improved?

Reliability and stability (N)

- Did you have any technical problems entering game content?
- Did you lose any entered data due to technical problems?
- How could we improve reliability and stability?

Delivering and updating games (N)

- Did you adjust the game or any game components while students were already playing?
- Did you experience any problems?
- Could it be improved?

General questions for components

- Did you use this component?
- Did you have any trouble using the component?
- What did you miss working with the component?
- How could we improve the component?
- Did you use the preview option for this component?
- What turned out to be handy when using the component?
- Do you feel comfortable using the component?

Development process

No questions prepared.

Summary

General design question

Scenario-based serious games are a specific type of digital games aimed at learning. A scenario extensively describes an interactive narrative in which learners carry out assignments in an environment that closely resembles professional practice, such as a law firm or a psychological clinic, in order to acquire professional competences. Assignments mostly involve ill-defined problems that often allow for multiple solutions requiring application of specific methodologies or tools, or collaboration with fellow learners. This type of games is used to acquire complex cognitive skills that involve higher-order activities like problem solving, reasoning, taking decisions or reflecting. Higher education increasingly calls for acquiring this type of skills.

The application of serious games may improve effectiveness and efficiency of education. Challenging assignments may increase learners' engagement and motivation, which results in better performance and possibly less dropout. In-game guidance will reduce necessary human guidance. To foster learning, the learner should be in control, learning should be situated and authentic, and transfer of learned skills to practice should be supported.

Although the market for serious games is very promising and still growing, the application within online universities faces a number of issues that call for dedicated development and delivery environments. Smaller target groups and available budgets ask for efficient development. Specific development team members like educators and domain experts, and adaptation to the capacities of individual students ask for specific tooling. Game adaptation, evaluation and research ask for specific and extensive logging and analysis of student data.

These considerations lead to the general design question that underlies this thesis: *How to design and develop a generic platform for fast and flexible development and delivery of a wide variety of scenario-based serious games that enable complex cognitive skills acquisition?*

The platform to be developed should be generic, in the sense that online universities should be able to develop and deliver their own scenario-based serious games for various content domains and learning purposes. The platform should also integrate game development, delivery and playing in one system, and be relatively easy to extend.

Fast and flexible game development should be enabled by a user-friendly authoring environment that enables teachers to author games fast and independently, that way lowering the threshold for developing these type of games. The environment should also allow for games having multiple authors, and allow for preview and reuse of game content.

Fast and flexible game delivery should be enabled by an intuitive and immersive player environment that enables students to play games in authentic settings. Bug fixing of

already deployed games and student support in case of problems should be possible and easy.

We took the following main design steps to answer the general design question:

(i) We identified the *intended users* of such a platform, namely teachers, students, administrators and ICT developers;

(ii) We set up *requirements* for each intended user and for the platform itself. To promote broad use, the platform should be fast, flexible, reliable and stable, and usable on multiple operating systems;

(iii) Based on the requirements, we chose for a *web-based platform*, a multilayered platform architecture and the use of Open source software.

(iv) We identified five *platform roles* that should each have their own working environment. Teachers would either have a 'developer' role to author games or have a 'tutor' role to monitor students. Students would have a 'student' role. Administrators or ICT developers would either have an 'administrator' role to manage platform users and components or a 'run manager' role to manage game runs and assign students to them;

(v) We designed a *domain model* that includes all platform entities, such as components, games, runs and users, and their interdependencies. We decided to define and store all game content and progress as XML strings, which would allow for easy extension with new components, without the need to modify the domain model;

(vi) Based on our previous experience in developing serious games we identified an initial set of *platform components* that enables acquiring complex cognitive skills. A so called Script component enables assembling game script in order to adapt the player environment to the individual student;

(vii) To enable easy extension with components we devised a *generic component design* that allows for defining components' game content to be authored, game progress to be stored and to be adapted by game script; and

(viii) In order to avoid teachers having to program game script, we chose to have them *assemble game script* via popup dialogues.

After taking the main design steps we started implementing the platform, which we called EMERGO (in English EMERGE: Efficient Method for ExpeRiential Game-based Education). We subsequently implemented the domain model, the various role environments and the initial set of components, all of which demanded more detailed design decisions. To foster easy extension with new components, we decided to design a single editor for authoring, validation, previewing and testing of all components' game content. We also decided to design a single player environment that handles rendering and adaptation of platform components, events triggered by students and game script, and storage of students' progress for analysis during and after a game session.

Initial evaluations showed that we managed to design and develop a generic, sustainable and extendible platform that allows for fast and flexible development and delivery of a wide variety of scenario-based serious games.

The platform is indeed *generic*. A set of various components enables a wide variety of game scenarios to be authored, to be played and to be monitored as is demonstrated by 26 games developed for six content domains by several online universities. Different role environments integrate game development, delivery and playing in one system. In addition, the platform can be used for education as well as for research purposes.

The platform's *sustainability* is demonstrated by the facts that: (i) it has been used over the years to develop dozens of games for thousands of students in total; (ii) the number of components has increased from initially 12 to 30 at the moment; and (iii) the first games developed are still in use.

Extendibility is facilitated by: (i) the generic component design, which serves as a template for defining new components and their structure, properties, content and interdependencies; (ii) the flat structure of the components where dependencies are defined by relations rather than by a hierarchy; and (iii) the platform's multilayered architecture that neatly separates different responsibilities and processes.

The platform allows for *fast and flexible game development*. Teachers can use its authoring environment to author game content, can reuse already developed components and game content, and can preview and test a game at any time. The result is more efficient development as is indicated by a decrease in production time by a factor three to four as compared with values found before (about 25 to 30 hours of development for one hour of study instead of about 100 hours previously). The Script component allows for adaptation of the player environment to the individual student and for modification of its own working by switching parts of game script on or off. In addition, it is possible to modify already deployed games where modifications are immediately available to students, e.g., in case of bugs or step-by-step game roll-out.

The platform also allows for *fast and flexible game delivery*. Web-based delivery ensures fast and easy delivery of games and of updates of games and the platform itself. A single player environment enables teachers to offer their own challenging, authentic and adaptable environments to students. It also enables teachers to give feedback by sending an in-game email as a virtual game character, e.g., in case of poor or excellent student performance. Administrators may configure game runs of the same game differently, which allows for delivery of different game versions to different target groups, e.g., in case of scientific experiments. They may also adjust students' progress during a game run, which allows for immediately helping students in case of problems or for simultaneously putting a group of students into another state of play, e.g., a next level. Initial evaluations also showed that we managed to design and develop an intuitive

immersive player environment, as evaluated by students who were satisfied to very satisfied about its user interface.

First additional design question

We did not fully manage to design and develop a *user-friendly* authoring environment. Some teachers had trouble to use the Script component without help and all teachers found two components, the Script and Conversations components, difficult to use. This made us question how we could improve the usability of authoring environments for serious games in general, which led to the first additional design question that has been further researched for this thesis: *How to improve the usability of authoring environments for serious games?*

We conducted an in-depth qualitative study of the usability of the EMERGO authoring environment by applying interviews with game authors.

We found the usability of the authoring environment to be problematic with respect to its *understandability* and *learnability*, which is in line with findings for comparable environments that also showed shortcomings with respect to these aspects. We found the environment's *operability* to be somewhat problematic and its *functionality* and *reliability* to be positively valued.

Flaws in *usability* appear to be caused by a lack of guidance and support but probably are also caused by the inherent complexity of the platform's components, exposed to authors during authoring. Most components appear to have a low complexity, some have a medium complexity and one, the Script component, has a high complexity. Usability may be improved by adding guidance and support but usability improvements may hardly have any effect on the inherent component complexity. In addition, authoring of components is quite detailed and low level, where assembly of game script is close to programming. This makes the authoring environment on the one hand more powerful and flexible but on the other hand probably more difficult to comprehend, which may be another cause for its found problematic understandability and learnability.

Based on remarks of interviewees we established *guidelines to improve the usability of authoring environments* for serious games, with respect to understandability and learnability.

To improve *understandability* we recommend to:

(i) Simplify authoring by offering an intuitive user interface, which might be different for different kinds of authors;

(ii) Reduce complexity by offering two levels of input, basic for novices and advanced for experts;

(iii) Offer examples of scenarios, games, and game components and how they relate to each other, so authors better understand what to do;
(iv) Offer a preview option to preview entered content at any time, so authors better understand what they are doing; and
(v) Use clear terminology fitting authors' expectations.

To improve *learnability* we recommend to:
(i) Offer clear instruction and wizards to guide authors during the authoring process; and
(ii) Offer information on didactics and use of components, so novice authors can make a quick start.

These guidelines seem to be general and (partly) applicable to other dedicated types of authoring environments.

Second additional design question

Although the initial evaluations of the player environment showed that students were satisfied to very satisfied about its user interface, we did not evaluated its usability in detail and were curious if students were still satisfied and if not, how we could improve its usability. This made us question how we could improve the usability of player environments for serious games in general, which led to the second additional design question that has been further researched for this thesis: *How to improve the usability of player environments for serious games?*

We conducted a detailed study of the usability of the EMERGO player environment by applying quantitative and qualitative research methods on a group of students.

We found the *usability* of the player environment to be quite low. Quantitative analysis showed the usability to be between "ok" and "good", which is lower than found for web applications in general. In addition, the *operability* of the environment and the *understandability* of its various components were rated "sufficient". However, qualitative analysis showed that students were predominantly positive about the environment's operability, and relatively negative about its understandability and *user interface aesthetics* (pleasing and satisfying interaction for the user). They missed operating instructions and would like to operate in a more realistic, more dynamic and less structured environment where they are more in control. In addition, the environment's *functionality* was somewhat problematic. Its *reliability* was predominantly valued positively.

The found low *usability* may be caused by students' young age, probably more familiar with playing video games that provide dynamic player environments where they are in control. The games used for the evaluation may also have had teething problems and may have been too restrictive or not challenging, enjoyable or exciting enough. Although the platform allows for creating realistic, dynamic and open environments where

students are in control, game developers may have their reasons for not using this potential.

Based on remarks of respondents, we established *guidelines to improve the usability of player environments* for serious games:

(i) Match the interface of the environment to the expectations of its intended users as much as possible;
(ii) Offer a timeline to show user's progress in time and learning;
(iii) Use scoring to encourage the user to perform better;
(iv) Let users formulate their own questions using text or speech analysis;
(v) Use text analysis of reports to be able to give substantive feedback;
(vi) Present operating instructions at the right time;
(vii) Use sounds to support interaction between the user and the environment; and
(viii) Always present clear messages in case of errors or time-consuming processes.

Conclusions

The EMERGO platform allows for more efficient serious games development (general design question), which makes it suitable for serving smaller target groups that are often found in higher education, e.g., for a specific course in a specific content domain. The possibilities to preview and test games during development at any time and to author game script without programming facilitate specific development team members like educators and domain experts. Adaptation to the individual student, game evaluation and research on learning effects are facilitated by extensive logging of students' progress and data analysis using game script.

Due to its independence of operating systems and use of Open source software online universities should be able to easily install and use the platform. However, the platform is not widely used, although it was originally intended to be. Over the years the platform has been used for prolonged periods of time by three other institutes for online education in the Netherlands. Various other (inter)national institutes for education have come to know the platform of which some have used it in exploratory way.

A wider use of the platform and wider application of serious games could be promoted by future improvements regarding usability (additional design questions) based on the presented guidelines in this thesis and possible new extensions, like a scenario editor to better support authors in their game design and to simplify the authoring process, and a 'researcher' role to better support educational research and game evaluation.

The platform and its underlying ideas, architecture, evaluations and future development as presented in this thesis, can be expected to contribute to new development and research in the fields of serious games development, instructional design and online education.

Samenvatting

De algemene ontwerpvraag

Scenariogebaseerde serious games zijn een specifiek type digitale games dat gericht is op leren. Een scenario beschrijft uitgebreid een interactief verhaal waarin studenten opdrachten uitvoeren in een omgeving die sterk lijkt op de beroepspraktijk, zoals een advocatenkantoor of een psychologische kliniek, en waarbij het doel is om professionele competenties te verwerven. Opdrachten hebben meestal betrekking op vaag gedefinieerde problemen met vaak meerdere mogelijke oplossingen die de toepassing van specifieke methodologieën of hulpmiddelen, of samenwerking met medeleerlingen vereisen. Dit soort games wordt gebruikt om complexe cognitieve vaardigheden te verwerven waarmee activiteiten van hogere orde gemoeid zijn, zoals problemen oplossen, redeneren, beslissingen nemen of reflecteren. Het hoger onderwijs vraagt steeds meer om het verwerven van dit soort vaardigheden.

Het gebruik van serious games kan de effectiviteit en efficiëntie van het onderwijs verbeteren. Uitdagende opdrachten kunnen de betrokkenheid en motivatie van studenten verhogen, wat resulteert in betere prestaties en mogelijk minder uitval. In de game ingebouwde begeleiding zal benodigde menselijke begeleiding verminderen. Om het leren te bevorderen, zal de student de controle moeten hebben, moet het leren gesitueerd en authentiek zijn en moet de overdracht van geleerde vaardigheden naar de praktijk worden ondersteund.

Hoewel de markt voor serious games zeer veelbelovend is en nog steeds groeit, kampt de toepassing binnen online universiteiten met een aantal problemen die vragen om specifieke ontwikkel- en uitleveromgevingen. Kleinere doelgroepen en beschikbare budgetten vragen om efficiënte game-ontwikkeling. Voor het onderwijs specifieke game-ontwikkelaars, zoals docenten en domeinexperts, en adaptatie aan het niveau van individuele studenten vragen om specifieke gereedschappen. Game-adaptatie, evaluatie en onderzoek vragen om specifieke en uitgebreide opslag en analyse van studentgegevens.

Deze overwegingen leiden tot de algemene ontwerpvraag die ten grondslag ligt aan dit proefschrift: Hoe kan een generiek platform worden ontworpen en ontwikkeld voor snelle en flexibele ontwikkeling en uitlevering van een breed scala aan scenariogebaseerde serious games die worden gebruikt om complexe cognitieve vaardigheden te verwerven?

Het te ontwikkelen platform moet generiek zijn, in die zin dat online universiteiten hun eigen scenariogebaseerde serious games zouden moeten kunnen ontwikkelen en uitleveren, voor verschillende inhoudsdomeinen en met verschillende leerdoelen. Het platform moet ook de ontwikkeling, de uitlevering en het spelen van games in één systeem integreren en relatief eenvoudig uitbreidbaar zijn.

Snelle en flexibele game-ontwikkeling moet worden gefaciliteerd door een gebruiksvriendelijke auteursomgeving die docenten in staat stelt om games snel en onafhankelijk te ontwikkelen, waarmee de drempel wordt verlaagd voor het ontwikkelen van dit soort games. Games zouden meerdere auteurs moeten kunnen hebben en preview en hergebruik van game inhoud moet mogelijk zijn.

Snelle en flexibele game-uitlevering moet worden gefaciliteerd door een intuïtieve en meeslepende afspeelomgeving die studenten in staat stelt om games te spelen in authentieke omgevingen. Het verhelpen van fouten in reeds uitgeleverde games en ondersteuning van studenten in geval van problemen moet mogelijk en gemakkelijk zijn.

We hebben de volgende belangrijke ontwerpstappen gezet om de algemene ontwerpvraag te beantwoorden:

(i) We hebben de *beoogde gebruikers* van een dergelijk platform geïdentificeerd, namelijk docenten, studenten, beheerders en ICT-ontwikkelaars;

(ii) We hebben *eisen* opgesteld voor elke beoogde gebruiker en voor het platform zelf. Om breed gebruik te bevorderen, moet het platform snel, flexibel, betrouwbaar en stabiel zijn en bruikbaar op meerdere besturingssystemen;

(iii) Op basis van de eisen hebben we gekozen voor een *webgebaseerd platform*, een meerlagige platformarchitectuur en voor het gebruik van opensourcesoftware;

(iv) We hebben vijf *rollen* geïdentificeerd die ieder hun eigen werkomgeving zouden moeten krijgen. Docenten vervullen ofwel een 'ontwikkelaar' rol om games te ontwikkelen, ofwel een 'tutor' rol om studenten te monitoren. Studenten vervullen alleen een 'student' rol. Beheerders of ICT-ontwikkelaars vervullen ofwel een ‘beheerder’ rol om platform gebruikers en componenten te beheren, ofwel een 'runbeheerder' rol om game-runs te beheren en er studenten aan toe te wijzen;

(v) We hebben een *domeinmodel* ontworpen dat alle platform-entiteiten bevat, zoals componenten, games, runs en gebruikers, en hun onderlinge afhankelijkheden. We hebben besloten om alle game-inhoud en -voortgang te definiëren en op te slaan als XML-tekst, waardoor eenvoudig nieuwe componenten kunnen worden toegevoegd zonder dat het domeinmodel hoeft te worden aangepast;

(vi) Op basis van onze eerdere ervaring met de ontwikkeling van serious games hebben we een initiële set van *platformcomponenten* geïdentificeerd die kan worden gebruikt om het verwerven van complexe cognitieve vaardigheden mogelijk te maken. Een zogenaamde Script-component faciliteert het invoeren van gamescript dat de afspeelomgeving kan aanpassen voor de individuele leerling;

(vii) Om een eenvoudige uitbreiding met componenten mogelijk te maken, hebben we een *generiek componentontwerp* ontwikkeld waarmee de game-inhoud van een component kan worden gedefinieerd, en welke gamevoortgang moet worden bijgehouden en kan worden aangepast door gamescript; en

(viii) Om te voorkomen dat docenten gamescript zouden moeten programmeren, hebben we ervoor gekozen om hen *gamescript te laten assembleren* met behulp van pop-updialogen.

Na het zetten van de belangrijkste ontwerpstappen zijn we begonnen met de implementatie van het platform dat we EMERGO noemden (Efficiënte Methodiek voor ERvaringsGericht Onderwijs). We implementeerden achtereenvolgens het domeinmodel, de werkomgevingen voor de verschillende rollen en de initiële set van platformcomponenten, waarbij we meer gedetailleerde ontwerpbeslissingen namen. Om de uitbreiding met nieuwe componenten te faciliteren, besloten we om één enkele editor te ontwerpen voor het invoeren, valideren, previewen en testen van de game-inhoud van alle componenten. We besloten ook om één enkele afspeelomgeving te ontwerpen voor de afhandeling van de presentatie en adaptatie van platformcomponenten, van studentacties en gamescript, en van de opslag van de studentvoortgang voor analyse tijdens en na een game.

Uit de eerste evaluaties bleek dat we erin geslaagd zijn om een generiek, duurzaam en uitbreidbaar platform te ontwerpen en te ontwikkelen dat een snelle en flexibele ontwikkeling en uitlevering van een breed scala aan scenariogebaseerde serious games faciliteert.

Het platform is inderdaad *generiek*. Een set van diverse componenten faciliteert het invoeren, spelen en monitoren van een grote verscheidenheid aan gamescenario's, zoals blijkt uit 26 games die door verschillende online-universiteiten voor zes inhoudsdomeinen zijn ontwikkeld. Verschillende werkomgevingen integreren het ontwikkelen, uitleveren en spelen van games in één systeem. Daarnaast is het platform behalve voor onderwijs ook voor onderzoek te gebruiken.

De *duurzaamheid* van het platform blijkt uit de volgende feiten: (i) het is jarenlang gebruikt om tientallen games te ontwikkelen voor in totaal duizenden studenten; (ii) het aantal componenten is gestegen van aanvankelijk 12 naar 30 op dit moment; en (iii) de als eerste ontwikkelde games worden nog steeds gebruikt.

De *uitbreidbaarheid* van het platform wordt gefaciliteerd door: i) het generieke componentontwerp dat dient als een sjabloon voor het definiëren van nieuwe componenten en hun structuur, eigenschappen, inhoud en onderlinge afhankelijkheden; ii) de platte structuur van de componenten, waarbij afhankelijkheden worden gedefinieerd door relaties in plaats van door een hiërarchie; en iii) de meerlagige platformarchitectuur, die verschillende verantwoordelijkheden en processen netjes scheidt.

Het platform faciliteert *snelle en flexibele game-ontwikkeling*. Docenten gebruiken de auteursomgeving om game-inhoud in te voeren, kunnen reeds ontwikkelde componenten en game-inhoud hergebruiken, en kunnen een game op elk gewenst moment previewen en testen. Dit heeft geresulteerd in efficiëntere game-ontwikkeling, zoals blijkt

uit een afname van de productietijd met een factor drie tot vier in vergelijking met eerder gevonden waarden (ongeveer 25 tot 30 uur ontwikkeltijd voor één uur studietijd in plaats van ongeveer 100 uur voorheen). De Script-component faciliteert het aanpassen van de afspeelomgeving voor de individuele leerling en het aanpassen van zijn eigen werking door delen van gamescript te kunnen in- of uitschakelen. Daarnaast is het mogelijk om reeds uitgeleverde games aan te passen waarbij de aanpassingen direct beschikbaar zijn voor studenten, bijvoorbeeld bij onvolkomenheden of een stapsgewijze uitrol van de game.

Het platform faciliteert ook een *snelle en flexibele game-uitlevering*. Webgebaseerde uitlevering zorgt voor een snelle en eenvoudige uitlevering van games, en van updates van games en het platform zelf. De afspeelomgeving stelt docenten in staat om hun eigen uitdagende, authentieke en aanpasbare omgevingen aan te bieden aan studenten. Ook kunnen docenten binnen de game terugkoppeling geven door een e-mail te verzenden namens een virtuele persoon, bijvoorbeeld bij tegenvallende of uitstekende prestaties van studenten. Beheerders kunnen game-runs van eenzelfde game verschillend configureren, waardoor verschillende versies van een game aan verschillende doelgroepen kunnen worden uitgeleverd, bijvoorbeeld tijdens wetenschappelijke experimenten. Ook kunnen beheerders de voortgang van studenten aanpassen tijdens een game-run, waardoor studenten direct kunnen worden geholpen in geval van problemen of een groep studenten tegelijkertijd in een andere game-toestand kan worden gezet, bijvoorbeeld een volgend game-level. De eerste evaluaties toonden ook aan dat we erin geslaagd zijn om een intuïtieve en meeslepende afspeelomgeving te ontwerpen en te ontwikkelen, wat bleek uit het feit dat studenten tevreden tot zeer tevreden waren over de gebruikersinterface.

De eerste aanvullende ontwerpvraag

We zijn er niet volledig in geslaagd om een *gebruiksvriendelijke* auteursomgeving te ontwerpen en te ontwikkelen. Sommige docenten hadden moeite om de Script-component zonder hulp te gebruiken en alle docenten vonden twee componenten, de Script- en Gesprekken-component, moeilijk te gebruiken. We vroegen ons af hoe de gebruiksvriendelijkheid van auteursomgevingen voor serious games in het algemeen verbeterd zou kunnen worden, wat leidde tot de eerste aanvullende ontwerpvraag die nader onderzocht is voor dit proefschrift: *Hoe kan de gebruiksvriendelijkheid van auteursomgevingen voor serious games verbeterd worden?*

We hebben een diepgaande kwalitatieve studie naar de gebruiksvriendelijkheid van de EMERGO-auteursomgeving uitgevoerd door een aantal auteurs te interviewen.

De gebruiksvriendelijkheid van de auteursomgeving bleek problematisch wat betreft *begrijpelijkheid* en *leerbaarheid*, wat in lijn is met bevindingen voor vergelijkbare omge-

vingen die ook tekortkomingen vertoonden wat betreft deze aspecten. *Operabiliteit* bleek enigszins problematisch en *functionaliteit* en *betrouwbaarheid* werden positief gewaardeerd.

De problematische *gebruiksvriendelijkheid* blijkt te worden veroorzaakt door een gebrek aan begeleiding en ondersteuning van auteurs, maar waarschijnlijk ook door de inherente complexiteit van de platformcomponenten, waarmee auteurs tijdens het invoeren geconfronteerd worden. De complexiteit blijkt voor de meeste componenten laag te zijn, voor sommige gemiddeld en voor één, de Script-component, hoog. De gebruiksvriendelijkheid kan worden verbeterd door begeleiding en ondersteuning toe te voegen, maar een verbeterde gebruiksvriendelijkheid zal weinig invloed hebben op de inherente complexiteit van de componenten. Daarnaast wordt de game-inhoud op een vrij gedetailleerd en laag niveau ingevoerd, waarbij het assembleren van gamescript dicht tegen programmeren aan ligt. Dit maakt de auteursomgeving aan de ene kant krachtig en flexibel, maar aan de andere kant waarschijnlijk moeilijker te begrijpen, wat een andere oorzaak kan zijn voor de gevonden problematische begrijpelijkheid en leerbaarheid.

Op basis van de opmerkingen van geïnterviewden hebben we *richtlijnen* opgesteld *om de gebruiksvriendelijkheid van auteursomgevingen voor serious games te verbeteren*, wat betreft hun begrijpelijkheid en leerbaarheid.

Om de *begrijpelijkheid* te verbeteren bevelen we aan om:

(i) Het invoeren van game-inhoud te vereenvoudigen door een intuïtieve gebruikersinterface aan te bieden, die per type gebruiker kan verschillen;
(ii) De complexiteit te verminderen door twee niveaus van invoer aan te bieden, simpel voor beginners en geavanceerd voor gevorderden;
(iii) Voorbeelden te geven van scenario's, games en componenten en hun onderlinge relaties, zodat gebruikers beter begrijpen wat ze moeten doen;
(iv) Een preview-optie aan te bieden die het mogelijk maakt om op elk moment ingevoerde game-inhoud te previewen, zodat auteurs beter begrijpen wat ze doen; en
(v) Duidelijke terminologie te gebruiken die aansluit bij de verwachtingen van de gebruikers.

Om de *leerbaarheid* te verbeteren bevelen we aan om:

(i) Duidelijke instructies en wizards aan te bieden om gebruikers tijdens het invoeren te begeleiden; en
(ii) Informatie aan te bieden over de didactiek en het gebruik van de componenten, zodat beginnende gebruikers een snelle start kunnen maken.

Deze richtlijnen lijken algemeen en (deels) toepasbaar te zijn op andere specifieke typen auteursomgevingen.

De tweede aanvullende ontwerpvraag

De eerste evaluaties van de afspeelomgeving toonden aan dat studenten tevreden tot zeer tevreden waren over de gebruikersinterface. We hadden de gebruiksvriendelijkheid echter niet in detail geëvalueerd en waren benieuwd of studenten nog steeds tevreden waren en zo niet, hoe we de gebruiksvriendelijkheid zouden kunnen verbeteren. We vroegen ons af hoe de gebruiksvriendelijkheid van afspeelomgevingen voor serious games in het algemeen verbeterd zou kunnen worden, wat leidde tot de tweede aanvullende ontwerpvraag die nader onderzocht is voor dit proefschrift: *Hoe kan de gebruiksvriendelijkheid van afspeelomgevingen voor serious games verbeterd worden?*

We hebben een gedetailleerde studie naar de gebruiksvriendelijkheid van de EMERGO-afspeelomgeving uitgevoerd door zowel kwantitatieve als kwalitatieve onderzoeksmethoden toe te passen op een groep studenten.

De *gebruiksvriendelijkheid* van de afspeelomgeving bleek vrij laag te zijn. Kwantitatieve analyse toonde aan dat de gebruiksvriendelijkheid tussen "ok" en "goed" was, wat lager is dan gevonden voor webapplicaties in het algemeen. De *operabiliteit* van de omgeving en de *begrijpelijkheid* van zijn verschillende componenten werden als "voldoende" beoordeeld. Kwalitatieve analyse toonde echter aan dat studenten overwegend positief waren over de operabiliteit van de omgeving, en relatief negatief over de begrijpelijkheid en de *esthetiek van de gebruikersinterface* (die zorgt voor een aangename en bevredigende interactie voor de gebruiker). Ze misten bedieningsinstructies en zouden graag willen opereren in een meer realistische en dynamische, en minder gestructureerde omgeving waarin ze meer controle hebben. Daarnaast was de *functionaliteit* van de omgeving enigszins problematisch. De *betrouwbaarheid* werd overwegend positief gewaardeerd.

De gevonden lage *gebruiksvriendelijkheid* kan worden veroorzaakt door de jonge leeftijd van de studenten, waardoor ze waarschijnlijk meer vertrouwd waren met het spelen van videogames die een dynamische omgeving bieden waarin ze meer controle hebben. De voor de evaluatie gebruikte games kunnen ook kinderziekten hebben gehad en kunnen te beperkend of niet uitdagend, plezierig of opwindend genoeg zijn geweest. Hoewel het platform het creëren van realistische, dynamische en open omgevingen waarin studenten controle hebben faciliteert, kunnen game-ontwikkelaars hun redenen hebben om dit potentieel niet te gebruiken.

Op basis van de opmerkingen van respondenten hebben we *richtlijnen* opgesteld *om de gebruiksvriendelijkheid van afspeelomgevingen voor serious games te verbeteren*:

(i) Stem de interface van de omgeving zoveel mogelijk af op de verwachtingen van de beoogde gebruikers;
(ii) Biedt een tijdlijn om de voortgang van de gebruiker in tijd en leren te tonen;
(iii) Gebruik scores om de gebruiker te stimuleren om beter te presteren;

(iv) Laat gebruikers hun eigen vragen formuleren door gebruik te maken van tekst- of spraakanalyse;
(v) Gebruik tekstanalyse van rapporten om inhoudelijke terugkoppeling te kunnen geven;
(vi) Presenteer bedieningsinstructies op het juiste moment;
(vii) Gebruik geluiden om de interactie tussen een gebruiker en de omgeving te ondersteunen; en
(viii) Presenteer altijd duidelijke meldingen bij fouten of tijdrovende processen.

Conclusies

Het EMERGO-platform maakt een efficiëntere ontwikkeling van serious games mogelijk (algemene ontwerpvraag), waardoor het platform geschikt is voor kleinere doelgroepen die vaak in het hoger onderwijs te vinden zijn, bijvoorbeeld voor een specifieke cursus in een specifiek inhoudsdomein. De mogelijkheden om de games tijdens de ontwikkeling op elk gewenst moment te kunnen previewen en testen, en om gamescript te kunnen invoeren zonder te hoeven programmeren, faciliteren specifieke ontwikkelteamleden zoals docenten en domeinexperts. Adaptatie aan de individuele student, game-evaluatie en -onderzoek naar leereffecten worden gefaciliteerd door uitgebreide opslag van de voortgang van studenten en data analyse met behulp van gamescript.

Dankzij zijn onafhankelijkheid van besturingssystemen en het gebruik van opensource-software zouden online-universiteiten in staat moeten zijn om het platform eenvoudig te installeren en te gebruiken. Het platform wordt echter niet op grote schaal gebruikt, hoewel dat oorspronkelijk wel de bedoeling was. Door de jaren heen is het platform langdurig gebruikt door drie andere instellingen voor online-onderwijs in Nederland. Diverse andere (inter)nationale onderwijsinstellingen hebben kennis genomen van het platform waarvan sommige het verkennend hebben gebruikt.

Een breder gebruik van het platform en een bredere toepassing van serious games zouden kunnen worden bevorderd door toekomstige verbeteringen wat betreft gebruiksvriendelijkheid (aanvullende ontwerpvragen) op basis van de gepresenteerde richtlijnen en mogelijke nieuwe uitbreidingen, zoals een scenario-editor om auteurs beter te ondersteunen bij het ontwerpen van games en om het invoeren te vereenvoudigen, en een 'onderzoeker' rol om onderwijskundig onderzoek en game-evaluatie beter te ondersteunen.

Het platform en zijn achterliggende ideeën, architectuur, evaluaties en toekomstige ontwikkeling, zoals gepresenteerd in dit proefschrift, zullen naar verwachting bijdragen aan nieuwe ontwikkelingen en onderzoek op het gebied van de ontwikkeling van serious games, instructie-ontwerp en online onderwijs.

Dankwoord

Graag bedank ik iedereen die een rol heeft gespeeld bij de totstandkoming van dit proefschrift. Vele helpende handen hebben de route naar de top mogelijk gemaakt. Ik verontschuldig mij bij voorbaat dat ik niet iedereen kan noemen.

Allereerst wil ik mijn promotores Rob Koper en Hans Hummel bedanken. Zij hebben erop vertrouwd dat ik de top zou halen, ook als ik het zicht op de top even kwijt was. Rob heeft me de mogelijkheid geboden om dit proefschrift te schrijven en heeft vooral in het begin erg veel tijd en energie geïnvesteerd en dat waardeer ik zeer. Hans, mijn steun en toeverlaat, heeft altijd voor me klaar gestaan. Zijn veelvuldige en uitgebreide feedback wat betreft aanpak, opbouw, inhoud en Engels van de verschillende hoofdstukken, en wat betreft de statistische analyse was van onschatbare waarde en heeft me op het goede pad gehouden.
Daarnaast dank ik Saskia Brand-Gruwel en Jos van den Broek voor het in mij gestelde vertrouwen en voor het bieden van de personele en financiële randvoorwaarden om dit proefschrift te kunnen schrijven.
Dan wil ik mijn reisgenoot Hub Kurvers bedanken. Zonder zijn vele werk was dit proefschrift er niet geweest. Als mede-architect, -ontwikkelaar en -beheerder van het EMERGO-platform en zijn componenten, en niet te vergeten als klankbord, heeft hij zeer bijgedragen aan het bereiken van de top. Ook zijn steun tijdens het promotietraject was onmisbaar.

Ik wil tevens de vele anderen bedanken die ervoor hebben gezorgd dat het EMERGO-platform kon worden ontwikkeld, gebruikt, doorontwikkeld of geëvalueerd. Zij vervulden vele verschillende rollen zoals: manager, lid van een stuurgroep, projectleider, didactisch ontwerper, scenarioschrijver, interactieontwerper, inhoudsdeskundige, auteur van game-inhoud, expert, acteur, regisseur, cameraman, video-editor, geïnterviewde, onderzoeker, ICT-beheerder, docent of student.
Allereerst bedank ik mijn OUNL-collega's die hebben bijgedragen aan vele projecten: Wim Westera, Hans Hummel, Rob Nadolski, Henk van den Brink, Jeroen Storm en Hub Kurvers. Veel dank voor jullie inbreng en voor de goede en fijne samenwerking.
Specifiek voor hun bijdrage aan het EMERGO project bedank ik: collega's van de OUNL, met name Colin Tattersall, Ruud Hoefakker en Ron Cörvers, en medewerkers van de Universiteit Utrecht, met name Carel Dieperink.
Voor hun bijdrage aan het Skills Labs project bedank ik: collega's van de OUNL, met name Ansje Löhr, Angelique Lansu, Daisy Tysmans en Mimi Crijns, en medewerkers van Hogeschool Zeeland en Kennis Netwerk Delta Water, met name Tony van der Hiele, Jasper van Houcke en Tjeerd Blauw.
Voor hun bijdrage aan de game-ontwikkeling bij de OUNL-cursus Seksuologie bedank ik: collega's van de OUNL, met name Henk van den Brink, Arjan Bos en Perry Pintar, en freelancer Mark Handels.
Voor hun deelname aan de interviews bedank ik: Henk van den Brink en Hub Kurvers.

Voor hun bijdrage aan het SLEM-project bedank ik: collega's van de OUNL, medewerkers van Stichting Praktijkleren, met name Ton Remeeus, Martin van Kollenburg en Mark Hector, en docenten en studenten van de vier deelnemende ROC's: Aventus, Nova College, roc van Twente en Zadkine.
Daarnaast dank ik iedereen die heeft bijgedragen aan een groot aantal niet in dit proefschrift beschreven projecten, die ofwel onderwijs ofwel onderzoek als oogmerk hadden. Meer specifiek bedank ik iedereen die heeft meegewerkt aan de productie van de vele uren video die hebben bijgedragen aan het succes van het platform.
Ik dank ook mijn OUNL-collega's die meer op de achtergrond het nodige werk hebben verricht. Bij het beheer van het platform was de bijdrage van medewerkers van GSO essentieel. Ook de bijdrage van medewerkers van de faculteit PenOW was essentieel. Zij hebben gefungeerd als inhoudsdeskundige, expert, acteur, of adviseur, of speelden een rol op administratief of secretarieel gebied. Meer specifiek bedank ik Mieke Haemers voor haar steun bij alle formaliteiten rondom de afronding van het promotietraject en de laatste correcties van het proefschrift, en Jeroen Berkhout voor de opmaak van de figuren in het proefschrift.

In het bijzonder wil ik Hans Hummel, Rob Nadolski en Hub Kurvers bedanken. Jullie zijn van meet af aan nauw betrokken geweest bij de ontwikkeling van het EMERGO-platform. Het is dan ook geen toeval dat Hans mijn dagelijks begeleider is geworden en Rob en Hub mijn paranimfen. Dank jullie, jongens, voor jullie inzet, steun en kameraadschap. Rob en Hans, deze laatste Tour zit erop, maar met jullie wil ik er nog wel de Giro of de Vuelta rijden.

Ik wil ook mijn collega's, familie, vrienden en kennissen bedanken voor de getoonde belangstelling, hun luisterend oor en hun goede raad. Een promotietraject vreet energie, maar gelukkig kon ik mijn accu altijd weer opladen bij de kookclub, muziekgroep Multivocaal, bij het meditatief zingen, bij de Djembe Folas en bij mijn tennismaten. Lieve mensen, dank hiervoor.

En natuurlijk wil ik ook mijn gezin bedanken. Zij hebben mij tijdens het promotietraject met hun liefde, adviezen en aanwezigheid gesteund en waren getuige van mijn pieken en dalen, en van mijn verminderde aanwezigheid. Vanaf nu ben ik er weer voor 100%!

Heerlen, juli 2018

Het werk gepresenteerd in hoofdstuk 2 en 3 is medegefinancieerd door stichting SURF, de ICT-samenwerkingsorganisatie van het onderwijs en onderzoek in Nederland (https://www.surf.nl/).

SIKS dissertation series

The complete list of dissertations carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems from 1998 on can be found at http://www.siks.nl/dissertations.php.

2011-01 Botond Cseke (RUN)
Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier (UU)
Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE)
Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU)
Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU)
Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE)
Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT)
Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU)
BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU)
Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT)
Cloud Content Contention

2011-11 Dhaval Vyas (UT)
Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE)
Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT)
Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR)
Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA)
The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM)
Selective Search in Games of Different Complexity

2011-17 Jiyin He (UVA)
Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM)
Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU)
The Mind ' s Eye on Personal Profiles

2011-20 Qing Gu (VU)
Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD)
Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UVA)
System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UVA)
Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT)
Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU)
Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU)
Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU)
Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein (UVA)
Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE)
Discrimination-aware Classification

2011-30 Egon van den Broek (UT)
Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR)
Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR)
Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU)
Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU)
Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU)
Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU)
Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN)
Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM)
Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU)
Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU)
Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT)
Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU)
Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU)
Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT)
Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT)
Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD)
Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz (VU)
Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT)
Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT)
Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012-01 Terry Kakeeto (UvT)
Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair (VU)
Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU)
Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU)
Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU)
Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU)
Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU)
When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

2012-08 Gerben de Vries (UVA)
Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT)
Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE)
Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE)
Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

2012-12 Kees van der Sluijs (TUE)
Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT)
Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

2012-14 Evgeny Knutov (TUE)
Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU)
Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

2012-16 Fiemke Both (VU)
Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT)
Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU)
Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE)
What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN)
Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD)
Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT)
Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT)
Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT)
Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT)
Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UVA)
Making Sense of Legal Text

2012-27 Hayrettin Gurkok (UT)
Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT)
Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT)
Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD)
Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN)
A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD)
Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN)
Coalitions in Cooperation Networks (COCOON)

2012-34 Pavol Jancura (RUN)
Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU)
Never Too Old To Learn -- On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN)
Analysis and Evaluation of Collaborative Modeling Processes

2012-37 Agnes Nakakawa (RUN)
A Collaboration Process for Enterprise Architecture Creation

2012-38 Selmar Smit (VU)
Parameter Tuning and Scientific Testing in Evolutionary Algorithms

2012-39 Hassan Fatemi (UT)
Risk-aware design of value and coordination networks

2012-40 Agus Gunawan (UvT)
Information Access for SMEs in Indonesia

2012-41 Sebastian Kelle (OU)
Game Design Patterns for Learning

2012-42 Dominique Verpoorten (OU)
Reflection Amplifiers in self-regulated Learning

2012-43 Withdrawn

2012-44 Anna Tordai (VU)
On Combining Alignment Techniques

2012-45 Benedikt Kratz (UvT)
A Model and Language for Business-aware Transactions

2012-46 Simon Carter (UVA)
Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

2012-47 Manos Tsagkias (UVA)
Mining Social Media: Tracking Content and Predicting Behavior

2012-48 Jorn Bakker (TUE)
Handling Abrupt Changes in Evolving Time-series Data

2012-49 Michael Kaisers (UM)
Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

2012-50 Steven van Kervel (TUD)
Ontologogy driven Enterprise Information Systems Engineering

2012-51 Jeroen de Jong (TUD)
Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013-01 Viorel Milea (EUR)
News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI)
MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

2013-03 Szymon Klarman (VU)
Reasoning with Contexts in Description Logics

2013-04 Chetan Yadati (TUD)
Coordinating autonomous planning and scheduling

2013-05 Dulce Pumareja (UT)
Groupware Requirements Evolutions Patterns

2013-06 Romulo Goncalves (CWI)
The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UvT)
Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk (VU)
Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN)
Metagenomic Data Analysis: Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige (UvT)
A Unified Modeling Framework for Service Design.

2013-11 Evangelos Pournaras (TUD)
Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Marian Razavian (VU)
Knowledge-driven Migration to Services

2013-13 Mohammad Safiri (UT)
Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

2013-14 Jafar Tanha (UVA)
Ensemble Approaches to Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM)
Multiagent Learning - Dynamic Games and Applications

2013-16 Eric Kok (UU)
Exploring the practical benefits of argumentation in multi-agent deliberation

2013-17 Koen Kok (VU)
The PowerMatcher: Smart Coordination for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT)
Outlier Selection and One-Class Classification

2013-19 Renze Steenhuizen (TUD)
Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UvA)
Fast and Reliable Online Learning to Rank for Information Retrieval

2013-21 Sander Wubben (UvT)
Text-to-text generation by monolingual machine translation

2013-22 Tom Claassen (RUN)
Causal Discovery and Logic

2013-23 Patricio de Alencar Silva (UvT)
Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM)
Automated Transfer in Reinforcement Learning

2013-25 Agnieszka Anna Latoszek-Berendsen (UM)
Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT)
Architectural Support for Dynamic Homecare Service Provisioning

2013-27 Mohammad Huq (UT)
Inference-based Framework Managing Data Provenance

2013-28 Frans van der Sluis (UT)
When Complexity becomes Interesting: An Inquiry into the Information eXperience

2013-29 Iwan de Kok (UT)
Listening Heads

2013-30 Joyce Nakatumba (TUE)
Resource-Aware Business Process Management: Analysis and Support

2013-31 Dinh Khoa Nguyen (UvT)
Blueprint Model and Language for Engineering Cloud Applications

2013-32 Kamakshi Rajagopal (OUN)
Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development

2013-33 Qi Gao (TUD)
User Modeling and Personalization in the Microblogging Sphere

2013-34 Kien Tjin-Kam-Jet (UT)
Distributed Deep Web Search

2013-35 Abdallah El Ali (UvA)
Minimal Mobile Human Computer Interaction

2013-36 Than Lam Hoang (TUe)
Pattern Mining in Data Streams

2013-37 Dirk Börner (OUN)
Ambient Learning Displays

2013-38 Eelco den Heijer (VU)
Autonomous Evolutionary Art

2013-39 Joop de Jong (TUD)
A Method for Enterprise Ontology based Design of Enterprise Information Systems

2013-40 Pim Nijssen (UM)
Monte-Carlo Tree Search for Multi-Player Games

2013-41 Jochem Liem (UVA)
Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning

2013-42 Léon Planken (TUD)
Algorithms for Simple Temporal Reasoning

2013-43 Marc Bron (UVA)
Exploration and Contextualization through Interaction and Concepts

2014-01 Nicola Barile (UU)
Studies in Learning Monotone Models from Data

2014-02 Fiona Tuliyano (RUN)
Combining System Dynamics with a Domain Modeling Method

2014-03 Sergio Raul Duarte Torres (UT)
Information Retrieval for Children: Search Behavior and Solutions

2014-04 Hanna Jochmann-Mannak (UT)
Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation

2014-05 Jurriaan van Reijsen (UU)
Knowledge Perspectives on Advancing Dynamic Capability

2014-06 Damian Tamburri (VU)
Supporting Networked Software Development

2014-07 Arya Adriansyah (TUE)
Aligning Observed and Modeled Behavior

2014-08 Samur Araujo (TUD)
Data Integration over Distributed and Heterogeneous Data Endpoints

2014-09 Philip Jackson (UvT)
Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language

2014-10 Ivan Salvador Razo Zapata (VU)
Service Value Networks

2014-11 Janneke van der Zwaan (TUD)
An Empathic Virtual Buddy for Social Support

2014-12 Willem van Willigen (VU)
Look Ma, No Hands: Aspects of Autonomous Vehicle Control

2014-13 Arlette van Wissen (VU)
Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

2014-14 Yangyang Shi (TUD)
Language Models With Meta-information

2014-15 Natalya Mogles (VU)
Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

2014-16 Krystyna Milian (VU)
Supporting trial recruitment and design by automatically interpreting eligibility criteria

2014-17 Kathrin Dentler (VU)
Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability

2014-18 Mattijs Ghijsen (UVA)
Methods and Models for the Design and Study of Dynamic Agent Organizations

2014-19 Vinicius Ramos (TUE)
Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support

2014-20 Mena Habib (UT)
Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

2014-21 Kassidy Clark (TUD)
Negotiation and Monitoring in Open Environments

2014-22 Marieke Peeters (UU)
Personalized Educational Games - Developing agent-supported scenario-based training

2014-23 Eleftherios Sidirourgos (UvA/CWI)
Space Efficient Indexes for the Big Data Era

2014-24 Davide Ceolin (VU)
Trusting Semi-structured Web Data

2014-25 Martijn Lappenschaar (RUN)
New network models for the analysis of disease interaction

2014-26 Tim Baarslag (TUD)
What to Bid and When to Stop

2014-27 Rui Jorge Almeida (EUR)
Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty

2014-28 Anna Chmielowiec (VU)
Decentralized k-Clique Matching

2014-29 Jaap Kabbedijk (UU)
Variability in Multi-Tenant Enterprise Software

2014-30 Peter de Cock (UvT)
Anticipating Criminal Behaviour

2014-31 Leo van Moergestel (UU)
Agent Technology in Agile Multiparallel Manufacturing and Product Support

2014-32 Naser Ayat (UvA)
On Entity Resolution in Probabilistic Data

2014-33 Tesfa Tegegne (RUN)
Service Discovery in eHealth

2014-34 Christina Manteli(VU)
The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.

2014-35 Joost van Ooijen (UU)
Cognitive Agents in Virtual Worlds: A Middleware Design Approach

2014-36 Joos Buijs (TUE)
Flexible Evolutionary Algorithms for Mining Structured Process Models

2014-37 Maral Dadvar (UT)
Experts and Machines United Against Cyberbullying

2014-38 Danny Plass-Oude Bos (UT)
Making brain-computer interfaces better: improving usability through post-processing.

2014-39 Jasmina Maric (UvT)
Web Communities, Immigration, and Social Capital

2014-40 Walter Omona (RUN)
A Framework for Knowledge Management Using ICT in Higher Education

2014-41 Frederic Hogenboom (EUR)
Automated Detection of Financial Events in News Text

2014-42 Carsten Eijckhof (CWI/TUD)
Contextual Multidimensional Relevance Models

2014-43 Kevin Vlaanderen (UU)
Supporting Process Improvement using Method Increments

2014-44 Paulien Meesters (UvT)
Intelligent Blauw. Met als ondertitel: Intelligence-gestuurde politiezorg in gebiedsgebonden eenheden.

2014-45 Birgit Schmitz (OUN)
Mobile Games for Learning: A Pattern-Based Approach

2014-46 Ke Tao (TUD)
Social Web Data Analytics: Relevance, Redundancy, Diversity

2014-47 Shangsong Liang (UVA)
Fusion and Diversification in Information Retrieval

2015-01 Niels Netten (UvA)
Machine Learning for Relevance of Information in Crisis Response

2015-02 Faiza Bukhsh (UvT)
Smart auditing: Innovative Compliance Checking in Customs Controls

2015-03 Twan van Laarhoven (RUN)
Machine learning for network data

2015-04 Howard Spoelstra (OUN)
Collaborations in Open Learning Environments

2015-05 Christoph Bösch (UT)
Cryptographically Enforced Search Pattern Hiding

2015-06 Farideh Heidari (TUD)
Business Process Quality Computation - Computing Non-Functional Requirements to Improve Business Processes

2015-07 Maria-Hendrike Peetz (UvA)
Time-Aware Online Reputation Analysis

2015-08 Jie Jiang (TUD)
Organizational Compliance: An agent-based model for designing and evaluating organizational interactions

2015-09 Randy Klaassen (UT)
HCI Perspectives on Behavior Change Support Systems

2015-10 Henry Hermans (OUN)
OpenU: design of an integrated system to support lifelong learning

2015-11 Yongming Luo (TUE)
Designing algorithms for big graph datasets: A study of computing bisimulation and joins

2015-12 Julie M. Birkholz (VU)
Modi Operandi of Social Network Dynamics: The Effect of Context on Scientific Collaboration Networks

2015-13 Giuseppe Procaccianti (VU)
Energy-Efficient Software

2015-14 Bart van Straalen (UT)
A cognitive approach to modeling bad news conversations

2015-15 Klaas Andries de Graaf (VU)
Ontology-based Software Architecture Documentation

2015-16 Changyun Wei (UT)
Cognitive Coordination for Cooperative Multi-Robot Teamwork

2015-17 André van Cleeff (UT)
Physical and Digital Security Mechanisms: Properties, Combinations and Trade-offs

2015-18 Holger Pirk (CWI)
Waste Not, Want Not! - Managing Relational Data in Asymmetric Memories

2015-19 Bernardo Tabuenca (OUN)
Ubiquitous Technology for Lifelong Learners

2015-20 Loïs Vanhée (UU)
Using Culture and Values to Support Flexible Coordination

2015-21 Sibren Fetter (OUN)
Using Peer-Support to Expand and Stabilize Online Learning

2015-22 Zhemin Zhu (UT)
Co-occurrence Rate Networks

2015-23 Luit Gazendam (VU)
Cataloguer Support in Cultural Heritage

2015-24 Richard Berendsen (UVA)
Finding People, Papers, and Posts: Vertical Search Algorithms and Evaluation

2015-25 Steven Woudenberg (UU)
Bayesian Tools for Early Disease Detection

2015-26 Alexander Hogenboom (EUR)
Sentiment Analysis of Text Guided by Semantics and Structure

2015-27 Sándor Héman (CWI)
Updating compressed colomn stores

2015-28 Janet Bagorogoza (TiU)
KNOWLEDGE MANAGEMENT AND HIGH PERFORMANCE; The Uganda Financial Institutions Model for HPO

2015-29 Hendrik Baier (UM)
Monte-Carlo Tree Search Enhancements for One-Player and Two-Player Domains

2015-30 Kiavash Bahreini (OU)
Real-time Multimodal Emotion Recognition in E-Learning

2015-31 Yakup Koç (TUD)
On the robustness of Power Grids

2015-32 Jerome Gard (UL)
Corporate Venture Management in SMEs

2015-33 Frederik Schadd (TUD)
Ontology Mapping with Auxiliary Resources

2015-34 Victor de Graaf (UT)
Gesocial Recommender Systems

2015-35 Jungxao Xu (TUD)
Affective Body Language of Humanoid Robots: Perception and Effects in Human Robot Interaction

2016-01 Syed Saiden Abbas (RUN)
Recognition of Shapes by Humans and Machines

2016-02 Michiel Christiaan Meulendijk (UU)
Optimizing medication reviews through decision support: prescribing a better pill to swallow

2016-03 Maya Sappelli (RUN)
Knowledge Work in Context: User Centered Knowledge Worker Support

2016-04 Laurens Rietveld (VU)
Publishing and Consuming Linked Data

2016-05 Evgeny Sherkhonov (UVA)
Expanded Acyclic Queries: Containment and an Application in Explaining Missing Answers

2016-06 Michel Wilson (TUD)
Robust scheduling in an uncertain environment

2016-07 Jeroen de Man (VU)
Measuring and modeling negative emotions for virtual training

2016-08 Matje van de Camp (TiU)
A Link to the Past: Constructing Historical Social Networks from Unstructured Data

2016-09 Archana Nottamkandath (VU)
Trusting Crowdsourced Information on Cultural Artefacts

2016-10 George Karafotias (VUA)
Parameter Control for Evolutionary Algorithms

2016-11 Anne Schuth (UVA)
Search Engines that Learn from Their Users

2016-12 Max Knobbout (UU)
Logics for Modelling and Verifying Normative Multi-Agent Systems

2016-13 Nana Baah Gyan (VU)
The Web, Speech Technologies and Rural Development in West Africa - An ICT4D Approach

2016-14 Ravi Khadka (UU)
Revisiting Legacy Software System Modernization

2016-15 Steffen Michels (RUN)
Hybrid Probabilistic Logics - Theoretical Aspects, Algorithms and Experiments

2016-16 Guangliang Li (UVA)
Socially Intelligent Autonomous Agents that Learn from Human Reward

2016-17 Berend Weel (VU)
Towards Embodied Evolution of Robot Organisms

2016-18 Albert Meroño Peñuela (VU)
Refining Statistical Data on the Web

2016-19 Julia Efremova (Tu/e)
Mining Social Structures from Genealogical Data

2016-20 Daan Odijk (UVA)
Context & Semantics in News & Web Search

2016-21 Alejandro Moreno Célleri (UT)
From Traditional to Interactive Playspaces: Automatic Analysis of Player Behavior in the Interactive Tag Playground

2016-22 Grace Lewis (VU)
Software Architecture Strategies for Cyber-Foraging Systems

2016-23 Fei Cai (UVA)
Query Auto Completion in Information Retrieval

2016-24 Brend Wanders (UT)
Repurposing and Probabilistic Integration of Data; An Iterative and data model independent approach

2016-25 Julia Kiseleva (TU/e)
Using Contextual Information to Understand Searching and Browsing Behavior

2016-26 Dilhan Thilakarathne (VU)
In or Out of Control: Exploring Computational Models to Study the Role of Human Awareness and Control in Behavioural Choices, with Applications in Aviation and Energy Management Domains

2016-27 Wen Li (TUD)
Understanding Geo-spatial Information on Social Media

2016-28 Mingxin Zhang (TUD)
Large-scale Agent-based Social Simulation - A study on epidemic prediction and control

2016-29 Nicolas Höning (TUD)
Peak reduction in decentralised electricity systems - Markets and prices for flexible planning

2016-30 Ruud Mattheij (UvT)
The Eyes Have It

2016-31 Mohammad Khelghati (UT)
Deep web content monitoring

2016-32 Eelco Vriezekolk (UT)
Assessing Telecommunication Service Availability Risks for Crisis Organisations

2016-33 Peter Bloem (UVA)
Single Sample Statistics, exercises in learning from just one example

2016-34 Dennis Schunselaar (TUE)
Configurable Process Trees: Elicitation, Analysis, and Enactment

2016-35 Zhaochun Ren (UVA)
Monitoring Social Media: Summarization, Classification and Recommendation

2016-36 Daphne Karreman (UT)
Beyond R2D2: The design of nonverbal interaction behavior optimized for robot-specific morphologies

2016-37 Giovanni Sileno (UvA)
Aligning Law and Action - a conceptual and computational inquiry

2016-38 Andrea Minuto (UT)
MATERIALS THAT MATTER - Smart Materials meet Art & Interaction Design

2016-39 Merijn Bruijnes (UT)
Believable Suspect Agents; Response and Interpersonal Style Selection for an Artificial Suspect

2016-40 Christian Detweiler (TUD)
Accounting for Values in Design

2016-41 Thomas King (TUD)
Governing Governance: A Formal Framework for Analysing Institutional Design and Enactment Governance

2016-42 Spyros Martzoukos (UVA)
Combinatorial and Compositional Aspects of Bilingual Aligned Corpora

2016-43 Saskia Koldijk (RUN)
Context-Aware Support for Stress Self-Management: From Theory to Practice

2016-44 Thibault Sellam (UVA)
Automatic Assistants for Database Exploration

2016-45 Bram van de Laar (UT)
Experiencing Brain-Computer Interface Control

2016-46 Jorge Gallego Perez (UT)
Robots to Make you Happy

2016-47 Christina Weber (UL)
Real-time foresight - Preparedness for dynamic innovation networks

2016-48 Tanja Buttler (TUD)
Collecting Lessons Learned

2016-49 Gleb Polevoy (TUD)
Participation and Interaction in Projects. A Game-Theoretic Analysis

2016-50 Yan Wang (UVT)
The Bridge of Dreams: Towards a Method for Operational Performance Alignment in IT-enabled Service Supply Chains

2017-01 Jan-Jaap Oerlemans (UL)
Investigating Cybercrime

2017-02 Sjoerd Timmer (UU)
Designing and Understanding Forensic Bayesian Networks using Argumentation

2017-03 Daniël Harold Telgen (UU)
Grid Manufacturing; A Cyber-Physical Approach with Autonomous Products and Reconfigurable Manufacturing Machines

2017-04 Mrunal Gawade (CWI)
MULTI-CORE PARALLELISM IN A COLUMN-STORE

2017-05 Mahdieh Shadi (UVA)
Collaboration Behavior

2017-06 Damir Vandic (EUR)
Intelligent Information Systems for Web Product Search

2017-07 Roel Bertens (UU)
Insight in Information: from Abstract to Anomaly

2017-08 Rob Konijn (VU)
Detecting Interesting Differences:Data Mining in Health Insurance Data using Outlier Detection and Subgroup Discovery

2017-09 Dong Nguyen (UT)
Text as Social and Cultural Data: A Computational Perspective on Variation in Text

2017-10 Robby van Delden (UT)
(Steering) Interactive Play Behavior

2017-11 Florian Kunneman (RUN)
Modelling patterns of time and emotion in Twitter #anticipointment

2017-12 Sander Leemans (TUE)
Robust Process Mining with Guarantees

2017-13 Gijs Huisman (UT)
Social Touch Technology - Extending the reach of social touch through haptic technology

2017-14 Shoshannah Tekofsky (UvT)
You Are Who You Play You Are: Modelling Player Traits from Video Game Behavior

2017-15 Peter Berck, Radboud University (RUN)
Memory-Based Text Correction

2017-16 Aleksandr Chuklin (UVA)
Understanding and Modeling Users of Modern Search Engines

2017-17 Daniel Dimov (UL)
Crowdsourced Online Dispute Resolution

2017-18 Ridho Reinanda (UVA)
Entity Associations for Search

2017-19 Jeroen Vuurens (TUD)
Proximity of Terms, Texts and Semantic Vectors in Information Retrieval

2017-20 Mohammadbashir Sedighi (TUD)
Fostering Engagement in Knowledge Sharing: The Role of Perceived Benefits, Costs and Visibility

2017-21 Jeroen Linssen (UT)
Meta Matters in Interactive Storytelling and Serious Gaming (A Play on Worlds)

2017-22 Sara Magliacane (VU)
Logics for causal inference under uncertainty

2017-23 David Graus (UVA)
Entities of Interest--- Discovery in Digital Traces

2017-24 Chang Wang (TUD)
Use of Affordances for Efficient Robot Learning

2017-25 Veruska Zamborlini (VU)
Knowledge Representation for Clinical Guidelines, with applications to Multimorbidity Analysis and Literature Search

2017-26 Merel Jung (UT)
Socially intelligent robots that understand and respond to human touch

2017-27 Michiel Joosse (UT)
Investigating Positioning and Gaze Behaviors of Social Robots: People's Preferences, Perceptions and Behaviors

2017-28 John Klein (VU)
Architecture Practices for Complex Contexts

2017-29 Adel Alhuraibi (UVT)
From IT-BusinessStrategic Alignment to Performance: A Moderated Mediation Model of Social Innovation, and Enterprise Governance of IT

2017-30 Wilma Latuny (UVT)
The Power of Facial Expressions

2017-31 Ben Ruijl (UL)
Advances in computational methods for QFT calculations

2017-32 Thaer Samar (RUN)
Access to and Retrievability of Content in Web Archives

2017-33 Brigit van Loggem (OU)
Towards a Design Rationale for Software Documentation: A Model of Computer-Mediated Activity

2017-34 Maren Scheffel (OUN)
The Evaluation Framework for Learning Analytics

2017-35 Martine de Vos (VU)
Interpreting natural science spreadsheets

2017-36 Yuanhao Guo (UL)
Shape Analysis for Phenotype Characterisation from High-throughput Imaging

2017-37 Alejandro Montes García (TUE)
WiBAF: A Within Browser Adaptation Framework that Enables Control over Privacy

2017-38 Alex Kayal (TUD)
Normative Social Applications

2017-39 Sara Ahmadi (RUN)
Exploiting properties of the human auditory system and compressive sensing methods to increase noise robustness in ASR

2017-40 Altaf Hussain Abro (VUA)
Steer your Mind: Computational Exploration of Human Control in Relation to Emotions, Desires and Social Support For applications in human-aware support systems"

2017-41 Adnan Manzoor (VUA)
Minding a Healthy Lifestyle: An Exploration of Mental Processes and a Smart Environment to Provide Support for a Healthy Lifestyle

2017-42 Elena Sokolova (RUN)
Causal discovery from mixed and missing data with applications on ADHD datasets

2017-43 Maaike de Boer (RUN)
Semantic Mapping in Video Retrieval

2017-44 Garm Lucassen (UU)
Understanding User Stories - Computational Linguistics in Agile Requirements Engineering

2017-45 Bas Testerink (UU)
Decentralized Runtime Norm Enforcement

2017-46 Jan Schneider (OU)
Sensor-based Learning Support

2017-47 Yie Yang (TUD)
Crowd Knowledge Creation Acceleration

2017-48 Angel Suarez (OU)
Collaborative inquiry-based learning

2018-01 Han van der Aa (VUA)
Comparing and Aligning Process Representations

2018-02 Felix Mannhardt (TUE)
Multi-perspective Process Mining

2018-03 Steven Bosems (UT)
Causal Models For Well-Being: Knowledge Modeling, Model-Driven Development of Context-Aware Applications, and Behavior Prediction

2018-04 Jordan Janeiro (TUD)
Flexible Coordination Support for Diagnosis Teams in Data-Centric Engineering Tasks

2018-05 Hugo Huurdeman (UVA)
Supporting the Complex Dynamics of the Information Seeking Process

2018-06 Dan Ionita (UT)
Model-Driven Information Security Risk Assessment of Socio-Technical Systems

2018-07 Jieting Luo (UU)
A formal account of opportunism in multi-agent systems

2018-08 Rick Smetsers (RUN)
Advances in Model Learning for Software Systems

2018-09 Xu Xie (TUD)
Data Assimilation in Discrete Event Simulations

2018-10 Julienka Mollee (VUA)
Moving forward: supporting physical activity behavior change through intelligent technology

2018-11 Mahdi Sargolzaei (UVA)
Enabling Framework for Service-oriented Collaborative Networks

2018-12 Xixi Lu (TUE)
Using behavioral context in process mining

2018-13 Seyed Amin Tabatabaei (VUA)
Using behavioral context in process mining: Exploring the added value of computational models for increasing the use of renewable energy in the residential sector

2018-14 Bart Joosten (UvT)
Detecting Social Signals with Spatio-temporal Gabor Filters

2018-15 Naser Davarzani (UM)
Biomarker discovery in heart failure

2018-16 Jaebok Kim (UT)
Automatic recognition of engagement and emotion in a group of children

2018-17 Jianpeng Zhang (TUE)
On Graph Sample Clustering

2018-18 Henriette Nakad (UL)
De Notaris en Private Rechtspraak

2018-19 Minh Duc Pham (VU)
Emergent relational schemas for RDF

2018-20 Manxia Liu (RUN)
Time and Bayesian Networks

2018-21 Aad Slootmaker (OU)
EMERGO: a generic platform for authoring and playing scenario-based serious games


Open Universiteit
www.ou.nl