

A Web-Based Serious Game for Joint Training

MASTER THESIS

KIT - KARLSRUHE INSTITUTE OF TECHNOLOGY
FRAUNHOFER IOSB - FRAUNHOFER INSTITUTE OF OPTRONICS,
SYSTEM TECHNOLOGIES AND IMAGE EXPLOITATION

Alexander Gundermann

April 30, 2016

Main Advisor:
Co-Advisor:

Prof. Dr.-Ing. Jürgen Beyerer
Dipl.-Inf. Alexander Streicher

Abstract

A Web-Based Serious Game for Joint Training

As military systems and processes are becoming increasingly complex and thus more difficult to comprehend, large-scale exercises for NATO Joint Intelligence, Surveillance, and Reconnaissance (JISR) are regularly performed yet are often obstructed by insufficient staff preparedness. To address this concern, this thesis presents the design, prototypal implementation, and evaluation of a modular and web-based serious game called Exercise Trainer (EXTRA). EXTRA is a simulation game for business processes that embeds metaphors on real-world systems into the game to facilitate recognition and to increase initial motivation towards its use. It is built on top of the Phaser.io game engine and the ReactJS user interface library. As a result of a subjective value benefit analysis, this combination turned out to be most appropriate. In addition, by encapsulating the game world state and user actions in self-contained data structures, EXTRA offers great potential in terms of interoperability with process models, learning platforms, and game authoring tools. A pilot study verified the technical functionality, the learning effects, and the user acceptance of the game. As the participants were able to recall the employed systems and their interrelationships, these research findings suggest the feasibility and effectiveness of EXTRA's approach to train exercise participants.

Kurzfassung

Web-basiertes Serious Game für Training im Verbund

Da Militärsysteme und -prozesse zunehmend komplexer werden, veranstaltet die NATO regelmäßig groß angelegte Übungen für Joint Intelligence, Surveillance und Reconnaissance (JISR). Allerdings können diese Übungen aufgrund von ineffektiver Vorbereitung seitens des Personals oftmals nicht, wie ursprünglich geplant, vollumfänglich durchgeführt werden. Um dem entgegenzuwirken, präsentiert diese Arbeit das Design, eine prototypische Implementierung und eine Evaluation eines modularen und web-basierten Serious Games namens Exercise Trainer (EXTRA). EXTRA ist ein Simulationsspiel für Geschäftsprozesse, das Metaphern echter Systeme in das Spielgeschehen einbindet, um den Wiedererkennungswert und die Anfangsmotivation im Hinblick auf die Nutzung zu steigern. EXTRA verwendet Phaser.io als Spiel-Engine und ReactJS als Bibliothek für die Darstellung der grafischen Oberfläche. Diese Kombination hat sich im Rahmen einer subjektiven Nutzwertanalyse als die am besten geeignetste erwiesen. Außerdem beschreibt EXTRA sowohl den Zustand der Spielwelt als auch die Befehle der Nutzer als eigenständige Datenstrukturen, wodurch sich vielfältige Möglichkeiten hinsichtlich der Interoperabilität mit Prozessmodellen, Lernplattformen und Spiel-Editoren ergeben. Eine Pilotstudie bestätigte die technische Funktionalität, die Lerneffekte sowie die Nutzerakzeptanz des Spiels. Da sich die Studienteilnehmer an die eingesetzten Systeme und deren Zusammenwirken erinnern konnten, deuten die Ergebnisse dieser Arbeit auf die Effektivität und Realisierbarkeit des Spielkonzepts hin.

Statement of authorship

I hereby declare that I have produced this work by myself except the utilities known to the supervisor, that I have labeled all used utilities completely and detailed and that I have labeled all material that has been taken with or without modification from the work of others.

Karlsruhe, April 30, 2016

Alexander Gundermann

Contents

1	Introduction	1
1.1	Background	1
1.2	Motivation	4
1.3	Scope and Outline	4
2	State of the Art	7
2.1	Gamified Assistance Systems	7
2.2	Computer-Based Training in the Military	11
2.3	Pre-Training Tools	14
3	Fundamentals	17
3.1	Value Benefit Analysis	17
3.2	Assistance Systems	19
3.3	Digital Game-Based Learning	21
3.4	HTML5 Game Engines	23
3.5	Web Development	25
3.5.1	JavaScript	25
3.5.2	NodeJS	26
3.5.3	Webpack	27
3.5.4	React	28
4	Game Engine Analysis	31
4.1	Overview of Alternatives	32
4.2	Criteria Tree	33
4.3	Results	34
4.4	Discussion	36
5	Exercise Trainer	39
5.1	Learning Objectives	39
5.2	Game Concept	40
5.3	Architecture	43
5.3.1	Client-Server Model	44
5.3.2	Game World State	45
5.3.3	Game Client Architecture	46

5.4	Design and Implementation	47
5.4.1	Scenario Specification	48
5.4.2	Actions	49
5.4.3	World and User Interface	51
5.4.4	Server Implementation	52
5.5	Application	53
6	Evaluation	55
6.1	Scenario	56
6.2	Design and Procedure	58
6.3	Participants	60
6.4	Results	60
6.5	Discussion	62
7	Conclusion	65
	Bibliography	67
	List of Figures	75

Introduction

In the face of ever-evolving technology, military systems and processes are becoming increasingly complex, posing a variety of challenges. This is particularly the case for military coalitions such as the NATO. Therefore, assistance and learning tools are being developed to ease the handling of technical systems, and large-scale exercises are regularly being performed to assess system interoperability and the overall performance.

To maximize the effectiveness of these exercises, this thesis presents the design, implementation and evaluation of a modular game concept called *Exercise Trainer* (EXTRA) to better prepare participants in a playful and intrinsically motivating way. Furthermore, the results of a value benefit analysis to find the best suited HTML5 game engine are presented. The game concept was developed at the *Fraunhofer Institute of Optronics, System Technologies and Image Exploitation* (Fraunhofer IOSB) in advance of this thesis.

This introductory chapter begins with a more thorough description of the application domain. Section 1.2 on page 4 explains the motivation behind the solution in detail, and section 1.3 on page 4 provides the scope and outline of this thesis.

1.1 Background

To make timely and informed decisions, military commanders and other action-takers need to be supplied with refined, accurate, and up-to-date information. This is achieved by gathering information from various sources (e.g., ground troops, aircraft, or satellites), which is then analyzed, interpreted, and fused with other information to produce actionable intelligence. This process is called *Joint Intelligence, Surveillance and Reconnaissance* (JISR) [NAT16a]. It is essential to all military operations. The NATO Communications and Information Agency (NCI Agency) describes the JISR process as a cycle comprising the elements *Task, Collect, Process, Exploit, and Disseminate* (TCPED) [NAT16b] as shown in figs. 1.1 and 6.2 on the following page and on page 56.

The principles of ISR have been applied in warfare for centuries. However, in today's digital age, many of the activities are executed or at least supported by computer technology. Additionally, the individual components are linked together, resulting in complex and heterogeneous systems. While highly capable, these systems pose new challenges in terms of interoperability, security, operation, and configuration. In coalitions such as the NATO, these issues are further amplified as different systems from different parties or nations have to be integrated. An example of a JISR system can be seen in fig. 1.2 on the following page.



Figure 1.1: Elements of the JISR cycle (Source: [NAT16b]).

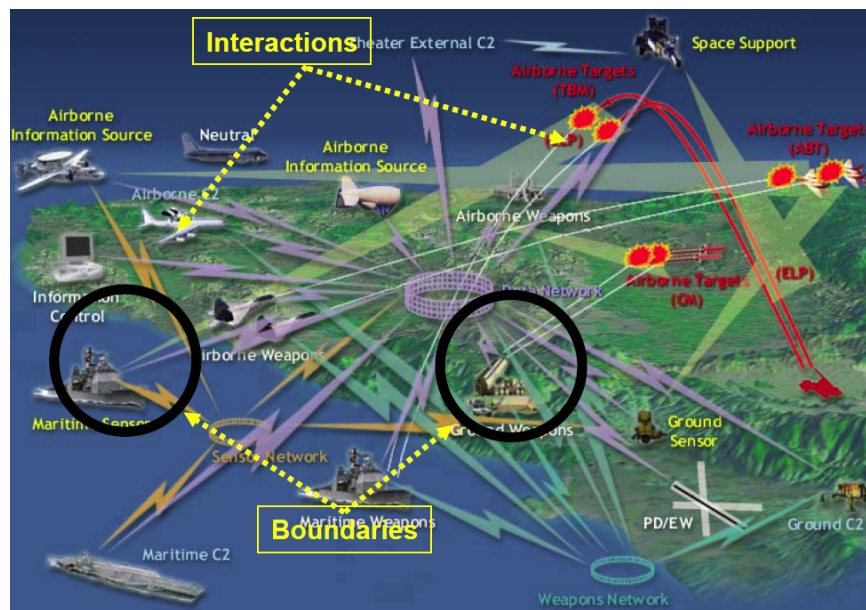


Figure 1.2: Exemplary model of a system of systems in the military (Source: [Fri06]).

To alleviate the issues of technical and procedural interoperability, the NATO has initiated a series of projects in recent years, the latest of which is called the *Multi-intelligence All Source Joint ISR Interoperability Coalition project* (MAJIIC 2) [Glo15]. MAJIIC 2 was the follow-up project of the original MAJIIC project and began in 2011. In total, nine nations took part, including Canada, France, Germany, the UK, and the US [Glo15]. Within the scope of these projects, several NATO *Standardization Agreements* (STANAGs) have been implemented and refined [Koc09]. The purpose of these agreements is to increase interoperability by specifying everything from data link formats to database configurations [Mar14].

To prove their interoperability architecture in practice, the NATO regularly performs large-scale, joint exercises in which these systems are deployed [NAT16a]. Such exercises involve many participants in different roles and with different backgrounds, who need to be aware of the processes and information flows related to the used systems. *Unified Vision 2014* (UV14), for example, included 2,000 participants from 18 NATO member countries [NAT14]. It featured an “array of surveillance

technology, equipment and personnel” and “a demanding operational environment to test the ability of [their] sensors, architecture and procedures to deliver intelligence capable of driving operations in the field” [NAT14]. It was preceded by UV12 in 2012, and is planned to be succeeded by UV16 in June 2016 [NAT16a].

Additionally, assistance systems are used to support operators and users of JISR systems by hiding the systems’ complexity and their technical aspects to make them easier to use. For example, the *Scenario Assistant* (SCENAS) was designed to automatically set up complex systems for training and demonstration purposes and to provide its users with an abstract, non-technical view of the system configuration [Str14]. Furthermore, e-learning systems, such as the *Mobile Assistant* (MOBAS) [Str13], can be used to train and educate operators in the face of ever-changing technology. Considering that today’s learners are accustomed to gaming [Pre05], e-learning systems can also incorporate game elements and mechanics (Gamification) to raise motivation and engagement [Mun11].

Another approach to utilize game concepts are educational games, i.e., games specifically designed for educational purposes. Educational games can be considered as a branch of serious games, which are more broadly defined as “games that do not have entertainment, enjoyment, or fun as their primary purpose” [Mic05].

Historically, the military has made extensive use of games for training and education: Chess and its predecessors, the origins of which date back thousands of years, were used to teach officers-in-training [Mic05]. In more recent times, the military started using computer-based simulators to train pilots, and video games for combat training and recruitment (e.g., America’s Army) [Mic05]. The application of game technology is not restricted to combat training, however; for example, the military also employs games such as Tactical Levantine and Tactical Iraqi (see fig. 1.3) to teach language and culture [Joh07, Mic05].



Figure 1.3: Screenshot of the game Tactical Iraqi (Source: [Hai10]).

1.2 Motivation

In 2012, the permanent representatives of the MAJIIC nations stated that “Operations in Afghanistan, and more recently in Libya, underlined several shortfalls in Alliance JISR processes”, one of them being “insufficient JISR dedicated staff preparedness” [Mar14]. Joint exercises such as UV14 seek to overcome these shortfalls by testing and optimizing the procedural and technical interoperability, coherence, and efficiency.

To ensure the efficiency of these exercises, the participants should bring at least a basic understanding of the systems involved in the scenario and their relationships. However, to date, no tools are available to prepare participants in this area. Since “a motivated learner can’t be stopped” [Pre03], it seems appropriate to follow a digital game-based learning approach to realize the preparation tool.

A computer-based solution also has the advantage of being able to adapt to different users [Str15], e.g., users speaking different languages and users seeking different levels of challenge. This approach also allows for content to be easily customized. For EXTRA, it is important to be able to adapt to different scenarios with little effort. In addition, a variety of devices can be supported to improve accessibility, including desktop computers and mobile devices.

Additionally, a web-based approach makes it possible to play the game from any device connected to the network on which the game is deployed (e.g., the internet) with just a web browser and without an installation process, thus further increasing accessibility. This approach also enables the game to be played in different scenarios [Rol13]. For example, the game can be used as a preparation tool using a desktop computer or a mobile device, but it could also be used in a more controlled environment as part of a blended learning scenario, where teachers can audit content, provide guidance, and observe the learners’ actions and progress.

Furthermore, social interactions, such as competition and collaboration, can be promoted using common game mechanics such as high score systems and multiplayer to further increase motivation. These mechanics can be realized using web technologies. Additionally, modern tools such as NW.js (nwjs.io) and Cordova (cordova.apache.org) are able to bundle web applications as native applications that do not require the device to be connected to the network and enable the use of native APIs, emphasizing the versatility of this approach.

1.3 Scope and Outline

The research questions of this thesis are twofold:

1. How to realize a web-based serious game to prepare participants of exercises involving complex, technical systems based on the given game-based concept?
2. Does the implementation fulfill its technical requirements and convey its learning contents?

The individual steps taken to address these questions are summarized in fig. 1.4 on the facing page and are reflected in the structure of this thesis.

The thesis starts with a **state-of-the-art analysis** in chapter 2 on page 7 to identify the work that has been performed in related fields and the concepts that have been employed. Since no existing preparation tool for exercises involving complex technical systems could be found, the chapter focuses on work in the fields of gamified assistance systems, games used in the military, and exercise preparation tools used for different purposes.

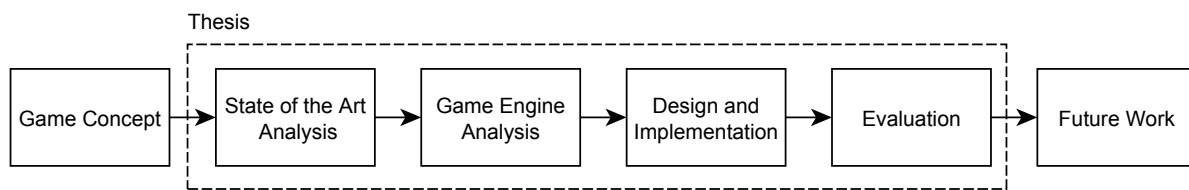


Figure 1.4: Scope and outline of the thesis.

Chapter 3 on page 17 summarizes the **theoretical background** and introduces modern **web technologies** relevant to the implementation of EXTRA.

A subjective **value benefit analysis** of HTML5 game engines precedes the implementation part and is presented in chapter 4 on page 31. The goal of the analysis is to systematically select one of the many available game engines best suited to fulfill the requirements of the game. To this end, each considered game engine has been rated according to a set of subjectively weighted criteria relevant to the game, yielding a final score to rank each engine.

The **implementation** is based on existing game concepts of an economic simulation game, using the previously selected game engine. The final result is a prototype of EXTRA. It enables players to take various perspectives (e.g., operators of specific components) in order to increase the understanding of the individual components and the system's overall functionality. To this end, JISR systems used in exercises can be embedded into the game on an abstract level and the game can be customized to adapt to different scenarios. Chapter 5 on page 39 describes the game's concepts and realization in detail.

Due to the complexity of developing a comprehensive digital game, however, some additional features often found in games (e.g., story, sounds, music, and appropriate user guidance) are left to be implemented. These requirements have been considered throughout the architecture and design of EXTRA. Advanced features, like online high scores and an engaging story line, have been examined and are subject to future work.

The **evaluation** of EXTRA presented in chapter 6 on page 55 is concerned with the technical functionality, the general usability and acceptance, as well as the game's effectiveness in conveying its learning contents regarding its concept. In addition to subjective data from questionnaires, this pilot user study includes data from recorded user actions. Due to time and resource constraints, an evaluation with end users was out of the scope of this thesis.

Finally, chapter 7 on page 65 provides a **summary** and proposes potential **future work** that has been identified throughout the thesis.

State of the Art

This chapter presents an overview of the state of the art in fields related to EXTRA. Since no existing tool for preparing participants of military exercises involving complex technical systems could be found, three related fields are examined: section 2.1 provides an overview of gamified assistance systems; section 2.2 on page 11 presents work in the vast field of computer-based training in the military, and section 2.3 on page 14 outlines some work related to pre-training in contexts different from EXTRA.

2.1 Gamified Assistance Systems

Assistance systems are a common solution to reduce the gap between human capabilities and technological potential [Wan05]. As assistance systems leave some tasks to be performed by the user, they often incorporate learning elements to make the overall task more efficient.

To further promote productivity and adoption, it is desirable for them to be engaging and motivating [Ree13]. As games exhibit these characteristics, there is a trend to integrate game elements into assistance systems. This process is typically called *gamification* [Det11]. Therefore, these systems are closely related to learning games such as EXTRA by providing assistance to use and understand complex systems.

The following sections summarize related work found in this field.

Production Environments

To establish gamification in assistance systems used in industrial production involving elderly and impaired workers, Korn, Funk, and Schmidt (2015) compared two implementations of gamified workplace systems and their non-gamified counterparts [Kor15]. Similar to EXTRA, their goal was to create a system with a high rate of acceptance to increase motivation and performance by using game design elements and providing context-aware assistance.

Their first system used an ordinary monitor and featured a complex visualization of the work steps, while the second system used projection and a simpler visualization as can be seen in fig. 2.1 on the following page. The second system resulted from findings of preceding research.

Both systems and their non-gamified counterparts were evaluated in real production environments. Whereas the evaluation of System 1 involved the assembly of Lego cars, the evaluation of System 2 was done with real products familiar to the workers.

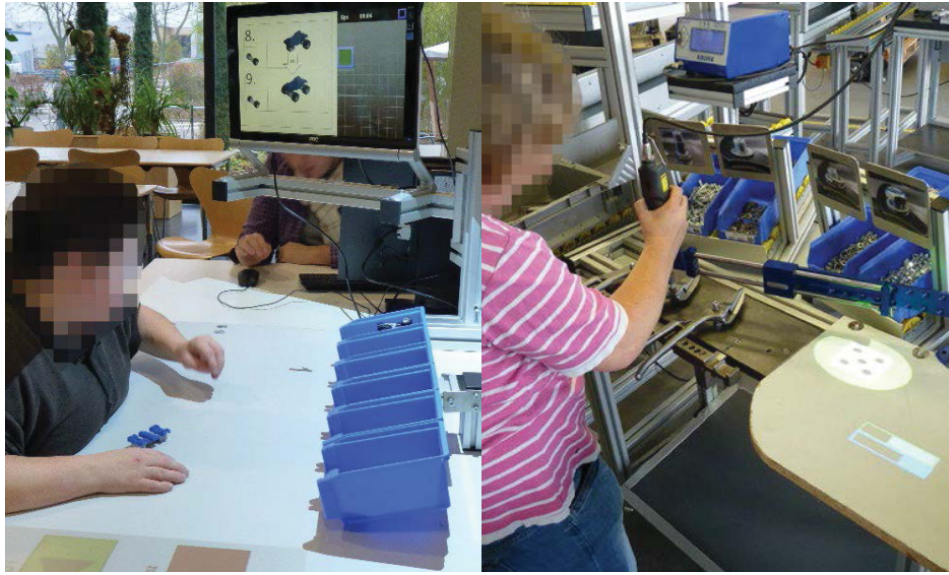


Figure 2.1: System 1 (left) uses a monitor to display gamification elements, and System 2 (right) uses projection and simpler visualizations (Source: [Kor15]).

The evaluations showed a slight increase in production rates for the gamified systems, but also an increase in error rates, suggesting an underlying trade-off. Based on the results of the *System Usability Scale* (SUS) questionnaire [Bro86], the acceptance was much higher for the second system which signifies the relevance of providing simple visualizations.

These findings may also be applicable to EXTRA in that the game should feature simple, understandable graphics to raise the perceived ease of use and thus user acceptance [Dav89]. Furthermore, the higher rate of errors underlines the importance of detecting user errors to provide feedback, as users seem to make more errors in gameful environments.

GamiCAD

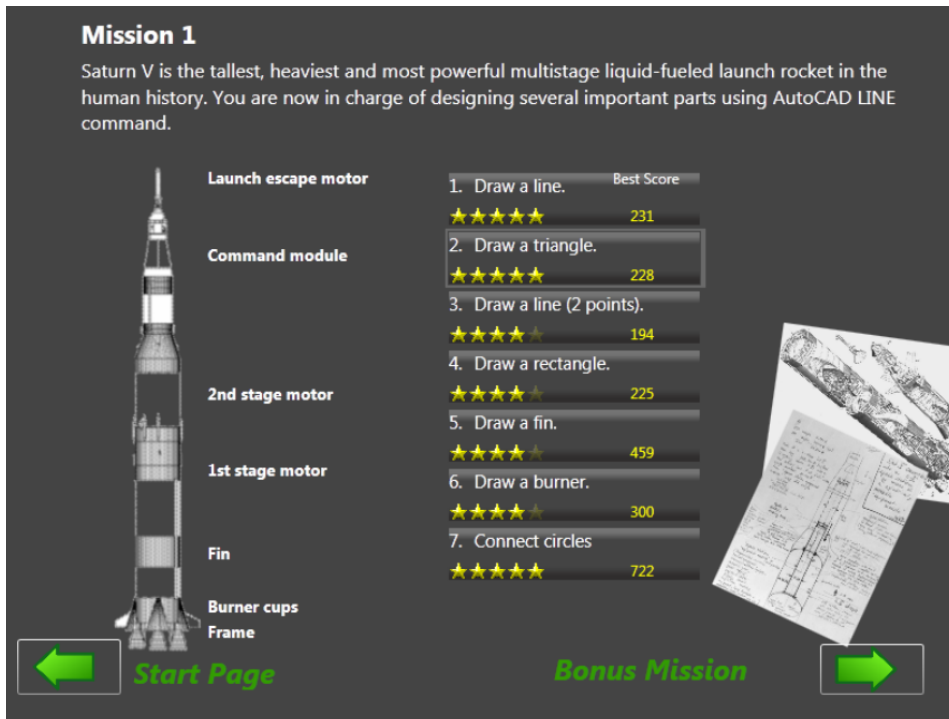
While EXTRA focuses on the understanding of complex technical systems in military scenarios, complex and difficult-to-use systems can also be found in the field of *Computer-Aided Design* (CAD) software. To provide a more engaging learning experience to new AutoCAD users, Wi, Grossmann, and Fitzmaurice (2012) presented the gamified tutorial system *GamiCAD* and conducted a user study to compare its effectiveness with a non-gamified solution [Li12].

In contrast to ordinary interactive tutorial systems, *GamiCAD* features unlockable levels, a score system based on completion time, several “arcade style” bonus levels, and a backstory involving the Apollo program. The level selection screen including a mission description and the high scores of each level can be seen in fig. 2.2 on the next page. Additionally, *GamiCAD* provides audiovisual feedback and helps users to recover from errors, which was realized using an event-driven finite state machine.

For the evaluation, gamification elements from *GamiCAD* were removed to create a second system, *TutorCAD* (see fig. 2.3 on the facing page), which could then be compared to *GamiCAD* in terms of user performance and subjective feedback. The results suggest higher subjective engagement levels

Mission 1

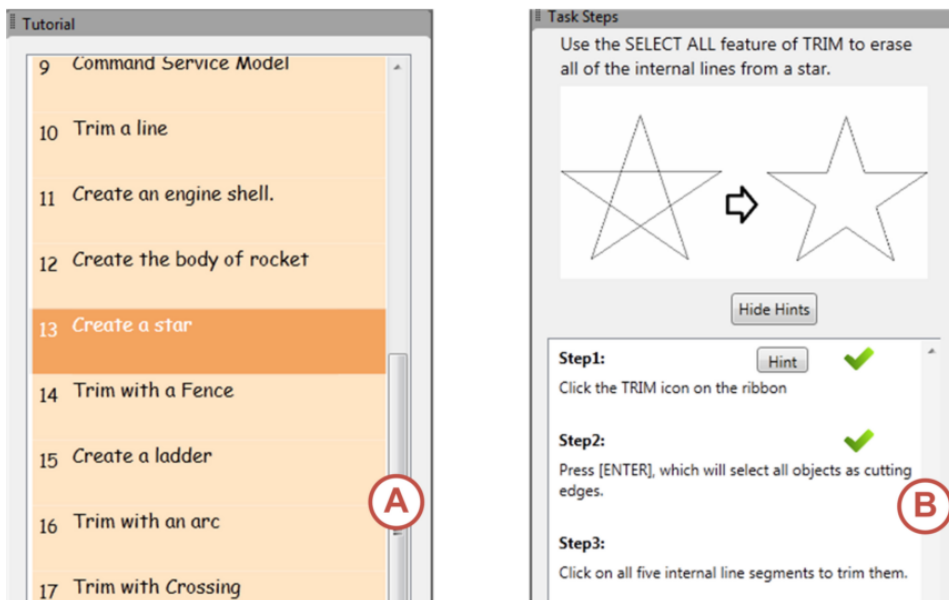
Saturn V is the tallest, heaviest and most powerful multistage liquid-fueled launch rocket in the human history. You are now in charge of designing several important parts using AutoCAD LINE command.



Task	Best Score
1. Draw a line.	231
2. Draw a triangle.	228
3. Draw a line (2 points).	194
4. Draw a rectangle.	225
5. Draw a fin.	459
6. Draw a burner.	300
7. Connect circles	722

Start Page Bonus Mission

Figure 2.2: Screenshot of the gamified mission page of GamiCAD (Source: [Li12]).



Tutorial

- 9 Command Service Model
- 10 Trim a line
- 11 Create an engine shell.
- 12 Create the body of rocket
- 13 Create a star
- 14 Trim with a Fence
- 15 Create a ladder
- 16 Trim with an arc
- 17 Trim with Crossing

Task Steps

Use the SELECT ALL feature of TRIM to erase all of the internal lines from a star.

Hide Hints

Step1: ✓
Click the TRIM icon on the ribbon

Step2: ✓
Press [ENTER], which will select all objects as cutting edges.

Step3:
Click on all five internal line segments to trim them.

Figure 2.3: Screenshots of the tutorial system without game elements (TutorCAD), showing the list of tutorials on the left and the task panel on the right (Source: [Li12]).

and an increased performance measured in task completion times and completion rates although some participants found the time pressure to be too stressful.

In the context of EXTRA, these results support the effectiveness of game-based approaches, but also show that the difficulty needs to correspond to the user's individual abilities to not frustrate the user.

SCENAS

To facilitate the setup and handling of complex systems for demonstration and training purposes, Streicher, Szentes, and Roller (2014) developed an assistance and e-learning system called *Scenario Assistant* (SCENAS) [Str14]. Similar to EXTRA, it provides the user with a simulation environment for learning, but it also offers building blocks to create scenarios for automatic configuration.

The goal of SCENAS is to eliminate the need for many specialists able to handle the individual subsystems. It was tested in a heterogeneous, interconnected system called *Experimental System for Image Exploitation* (ExBA), which comprises various user interfaces, sensor platforms, and processing algorithms.



Figure 2.4: Android Client of SCENAS (Source: [Str14]).

On an architectural level, SCENAS is built as a client-server system. Clients can connect to the server via HTTP to be presented with a HTML5 application which lets them view and start scenarios (see fig. 2.4). Every subsystem involved is connected to the server by an agent that acts as a mediator between SCENAS' representation format and the interface of the subsystem. These agents communicate with each other and the server via XMPP.

SCENAS also includes a subsystem called *Mobile Assistant* (MOBAS) [Str13] for learning and context-aware assistance in the installation, maintenance, and handling of complex systems. Building upon micro-learning concepts, MOBAS' learning units are small and self-contained. Additionally, MOBAS features playful elements in the form of quizzes to engage and motivate its users. Consequently, MOBAS and EXTRA are closely related in their application and in their approach of employing game elements.

2.2 Computer-Based Training in the Military

EXTRA is certainly not the first computer game designed to be used in a military context. Computer games and simulations are being used extensively in this field for a wide range of applications. Their main selling points are the cost savings and the interactive learning environment they can provide as opposed to instructor-lead presentations [Rom08].

As noted by Roman and Brown (2008), the emphasis on experience-based learning can be traced back to the cold-war era. At this time, training budgets were higher and thus more live training could be performed [Rom08]. Today's modern computer technologies offer a cost-effective alternative to this practice.

Still, various researchers have pointed out that such technology is most effective when used as part of a blended learning approach [Rom08, Fau14, Bre10]; that is, computer-based training should not entirely replace existing training and learning programs, but rather be carefully integrated.

In the following, two relevant works related to computer-based training in the military are presented.

Flooding Control Trainer

Hussain et al. (2009) developed a prototype training game called *Flooding Control Trainer* (FCT) for the U.S. Navy, sponsored by the Office of Naval Research [Hus09]. Their goal was to supplement classroom instructions for better training of recruits by realizing a training platform that can be used by different Navy technical schools and eventually in the fleet. They turned to serious games because of their potential to provide engaging learning environments at relatively low resource requirements.

Thus, their goals and approach are similar to those of EXTRA. In contrast, however, FCT is implemented as a first-person, single-player game set in a realistic 3D environment (see fig. 2.5), i.e., their solution does not try to abstract any real-world elements.



Figure 2.5: Screenshot of Flooding Control Trainer (Source: [Hus09]).

As the name of FCT suggests, the training goal is to teach basic skills to control flooding on board of a ship. During gameplay, the players are expected to develop a sound mental model for situational awareness, communication, and decision-making as well as extend their shipboard navigation skills.

The game is story-driven and features various missions including a tutorial. The players are continuously guided and receive feedback based on their actions. They can also consult the help system called Navypedia. The instructional logic can be edited using a visual logic editor for rapid prototyping and modifications.

Overall, Hussain et al. performed two studies to evaluate the effectiveness of FCT: a usability study and a validation study. They conducted both studies with U.S. Navy recruits, which is the target population of the game. The usability study included customized versions of the *Questionnaire for User Interface Satisfaction* (QUIS) [Chi88] and the *System Usability Scale* (SUS) [Bro86]. The results were mostly positive, but showed the need for a tutorial level to accustom players to the navigation controls, which was then implemented.

For the controlled validation study, the students' performances were evaluated in a real test scenario two days after the training intervention which comprised one hour of playing time. The control group received no additional training at all. The results showed that the treatment group made far less errors and were more confident during the test scenario.

These results are relevant to this thesis as they show the effectiveness of game-based interventions, although this finding seems rather narrow, since the control group did not receive training by means of other methods for comparison. The tutorial level they implemented as a result of their first study stresses the need for games to adequately introduce their users to the games' controls.

SAR-Tutor

While advancements in technology increase the complexity of system architecture and necessitate learning tools such as EXTRA and MOBAS (see section 2.1 on page 10), technological advancements also affect other fields such as the interpretation of radar images.

Since *Synthetic Aperture Radar* (SAR) images are inherently difficult to interpret due to unusual distorting effects not found in optical images, there is an increased need for professionals and training in this area [Sze08].

To address this concern, Szentes, Bargel, Berger, and Roller (2008) developed a software solution called *SAR-Tutor*, which facilitates and supports the training and education of radar image interpreters [Sze08]. SAR-Tutor is being used by the German armed forces (Bundeswehr). To avoid boredom and to provide a diverse learning experience, it includes an array of different exercise types, including multiple choice, drag and drop, and image exploitation exercises. One of these exercises can be seen in fig. 2.6 on the facing page.

Similar to EXTRA, one challenge they faced regarding the design of the software was to accommodate a heterogeneous group of users with varying educational backgrounds and ages. Therefore, their didactic approach assumed a minimum educational level, focusing on videos and interactive animations instead of complicated formulas.

Also, the content was arranged in freely navigable books and chapters to ease the handling of the software. They classified their approach as being constructivist, i.e., learners are encouraged to build up new knowledge while interacting with the system based on their existing knowledge and experiences.

Due to its server-based architecture and component-based design, SAR-Tutor can be used in different blended learning scenarios as illustrated in fig. 2.7 on the next page: It can be used alongside

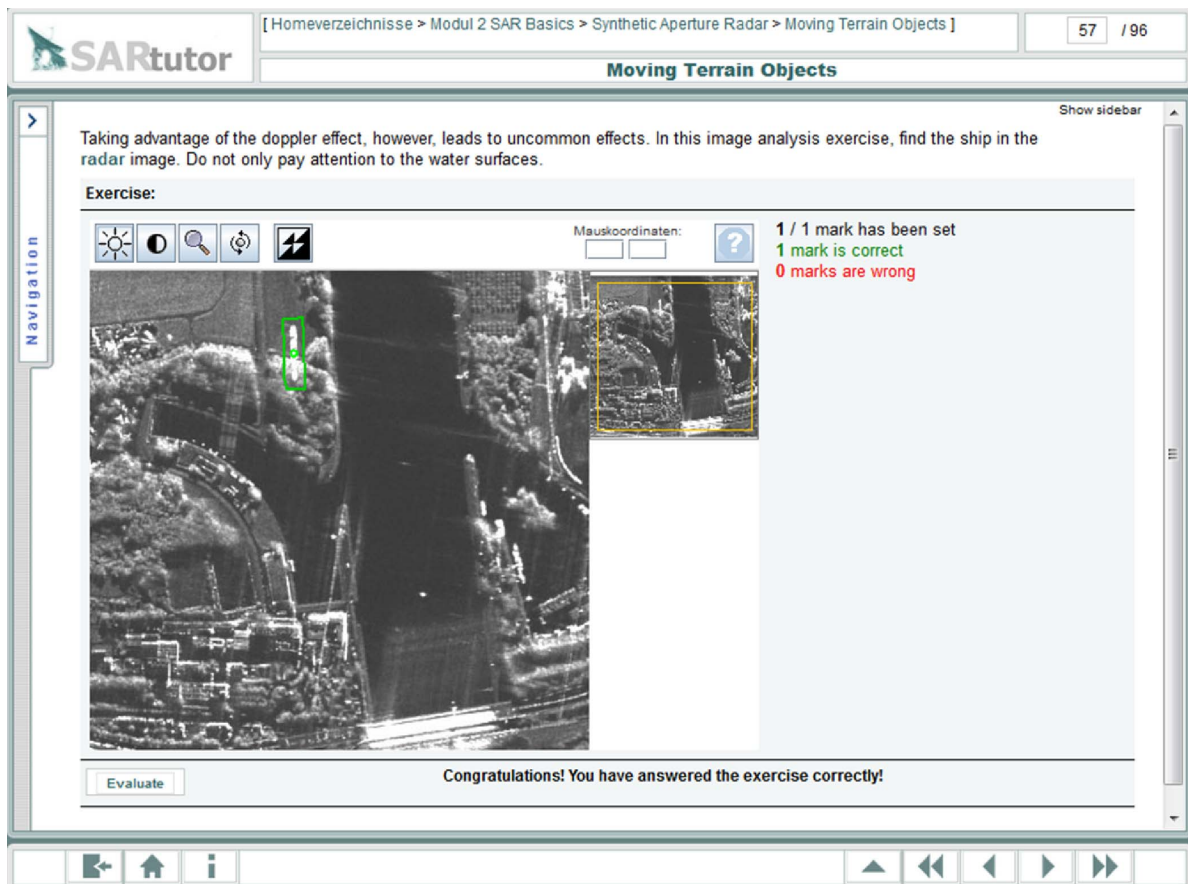


Figure 2.6: Screenshot of SAR-Tutor showing the evaluation of a task in which the user needed to spot and click on a ship bordered in green (Source: [Rol13]).

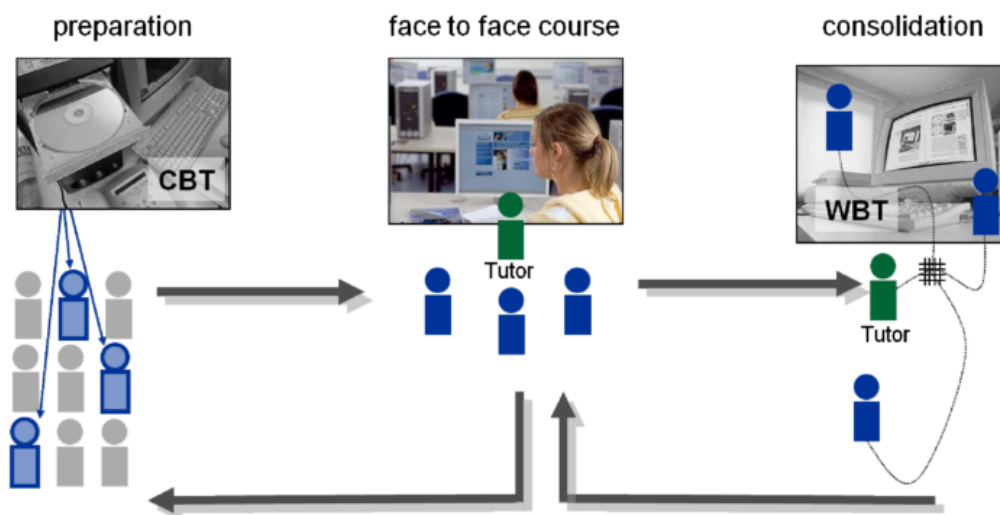


Figure 2.7: Different application scenarios of SAR-Tutor (Source: [Rol13]).

classroom lectures in training centers to deepen the imparted knowledge (*Computer-Based Training*, or CBT), or it can be used after the phase of attendance to consolidate knowledge or offer help during image interpretation. In the latter case, the content is centralized and managed by human tutors who are also able to communicate with the users (*Web-Based Training*, or WBT). Finally, SAR-Tutor can be used before the attendance phase, ideally leveling the knowledge of its heterogeneous users. Such versatility in terms of application is also desired for EXTRA.

In relation to SAR-Tutor, Roller, Berger, and Szentes (2013) concluded that more work needs to be done to intrinsically motivate learners and to make the learning system adaptive by monitoring user actions and behavior [Rol13]. In this regard, they proposed a game-based approach for the “new generation of interpreters” [Rol13]. Further, they suggested (online) contests based on scoring and ranking mechanisms for a more stimulating learning process. These ideas have been incorporated into the concept of EXTRA.

2.3 Pre-Training Tools

While the effectiveness of serious games is hard to assess, there appears to be consensus on the positive effects when used for pre-training learning interventions [Rom08]. This insight is not limited to serious games and seems to be applicable to other learning methods as well [May02, MM10]. Bowers et al. (2013) have found text-only pre-experiences to be ineffective in a military context, however [Bow13].

As these tools are related to EXTRA in their application, the following section provides a summary of work related to pre-training.

Multiplayer Virtual World for CPR training

Similar to this thesis, the work by Creutzfeldt, Hedman, and Felländer-Tsai (2012) shows the design and assessment of a serious game set in a virtual learning environment and used for pre-training [Cre12]. Their domain *Cardiopulmonary Resuscitation* (CPR) is a very different one, however. Their solution makes use of a multiplayer virtual world with avatars (see fig. 2.8) that allows players to interactively practice a range of scenarios.



Figure 2.8: Screenshot of the CPR pre-trainer by Creutzfeldt, Hedman, and Felländer-Tsai (Source: [Cre10]).

Their work was motivated by the fact that training in this field is still based on conventional methods including theoretical introductions and manikin training, despite the successful application of serious games in other fields of medicine.

For the assessment, they conducted an exploratory quasi-experimental controlled study which included three groups: one control group and two treatment groups. In addition to conventional training, the treatment groups underwent the virtual pre-training 6 and 18 months, respectively, before being assessed in a full-scale simulator. All 36 participants were medical students attending the same semester.

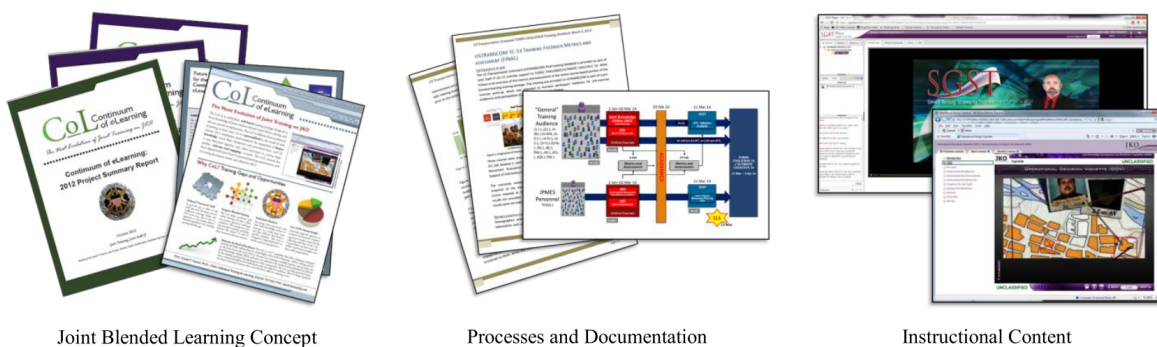
The main findings were that the pre-training was effective in increasing students' performance and knowledge, with the 6-month treatment group scoring 93% in the knowledge test as opposed to 65% for the control group. However, they questioned the validity of the performance assessment as it was done using a simulator.

As the students were still following a regular curriculum, these results underline the potential benefits of game-based interventions in terms of knowledge acquisition and retention in blended learning scenarios.

Blended Learning Training System

Based on three years of research and development from 2011 to 2014, Fautua, Schatz, Reitz, and Bockelman (2014) presented their approach to integrate a *Blended Learning Training System* (BLTS) into large-scale training exercises [Fau14]. The Joint Staff J7 of the U.S. Department of Defense initiated the project, and the project's goal was to overcome gaps related to training events, such as lack of attendance and knowledge retention.

Thus, their motivation, domain, and goals are closely related to those of EXTRA; their solution, however, is much broader in scope: for each individual training event, so-called *Blended Learning Training Packets* are created to align with the event's objectives and schedule. These packets include traditional online courses, simulation-based online training, and multimedia to augment live events. This instructional material is gathered in a repository and the processes for their development and execution are documented. As illustrated in fig. 2.9, the repository, documentation, and the concept itself are the three central components of the overall system.



Joint Blended Learning Concept

Processes and Documentation

Instructional Content

Figure 2.9: The three components of BLTS: (1) the blended learning concept, (2) processes and documentation, and (3) a repository of instructional content (Source: [Fau14]).

The BLTS was evaluated three times from 2012 to 2014: In 2012, a controlled study with surveys and tests was conducted during the Panamax Exercise, where the treatment group scored higher in knowledge tests after the exercise and reported to having been better prepared. During the Savannah Shield Exercise in 2013, the participation was suboptimal, yet trainers and trainees reacted positively according to surveys. Finally, in 2014, there was no statistically significant difference in the knowledge score, but, again, the treatment group felt slightly better prepared.

In the future, EXTRA could be integrated as part of such a system. Even though their results in terms of knowledge acquisition were mixed, the fact that exercise participants felt better prepared could indicate a high perceived usefulness of learning tools for exercise preparation, which in turn could result in a higher rate of acceptance [Dav89].

3

Fundamentals

This chapter provides the theoretical background of the concepts and technologies used in EXTRA. For this purpose, section 3.1 describes the method used for the game engine analysis presented in chapter 4 on page 31. Section 3.2 on page 19 and section 3.3 on page 21 explain the concepts of assistance systems and digital game-based learning, respectively, and illustrate their relation to EXTRA. Section 3.4 on page 23 explains the definition and general structure of HTML5 game engines. Finally, section 3.5 on page 25 presents various modern technologies used for the implementation.

3.1 Value Benefit Analysis

To locate the best suited game engine for the realization of EXTRA (see chapter 4 on page 31), this thesis uses a subjective *Value Benefit Analysis* (VBA). This approach is based on the game engine selection workflow proposed by Szentes, Bargel, and Streicher (2011) [Sze11].

In general, the purpose of the VBA is to score a (finite) set of alternatives in a systematic manner to create a ranking or to discover the best suited alternative. This kind of analysis is useful in complex situations which require balancing multiple factors or include subjective emotional factors (so-called soft factors) [Sze11]. There are various methods for determining the best decision in the field of operations research. The one used in this thesis is based on the *Simple Additive Weighting* (SAW) method or *Weighted Sum Method* (WSM), which is considered the simplest and most used approach of its class [Rao08].

The SAW method belongs to the discipline of *Multiple Criterion Decision-Making* (MCDM), which involves making decisions in the presence of multiple, usually conflicting, criteria [Rao08]. Moreover, MCDM methods can be further classified as either *Multiple Objective Decision-Making* (MODM) methods or *Multi-Attribute Decision-Making* (MADM) methods. The difference between the two is that MODM methods attempt to find an optimal solution given a set of goals and a large or infinite set of alternatives, while MADM methods are concerned with selecting the most suitable alternative from a pre-defined set based on their properties. Thus, SAW belongs to the category of MADM methods. Hwang and Yoon (1981) provide a detailed classification of these methods, which is demonstrated in fig. 3.1 on the next page [Hwa81].

The procedure of the VBA method can roughly be described as follows [Sze11]: first, a hierarchy of criteria and nested sub-criteria is established. This hierarchy should include all relevant criteria and should be as general as possible for the particular domain. Subsequently, each criterion (i.e., each node in the target tree) is assigned a weight (e.g., between 1 and 10) according to its significance.

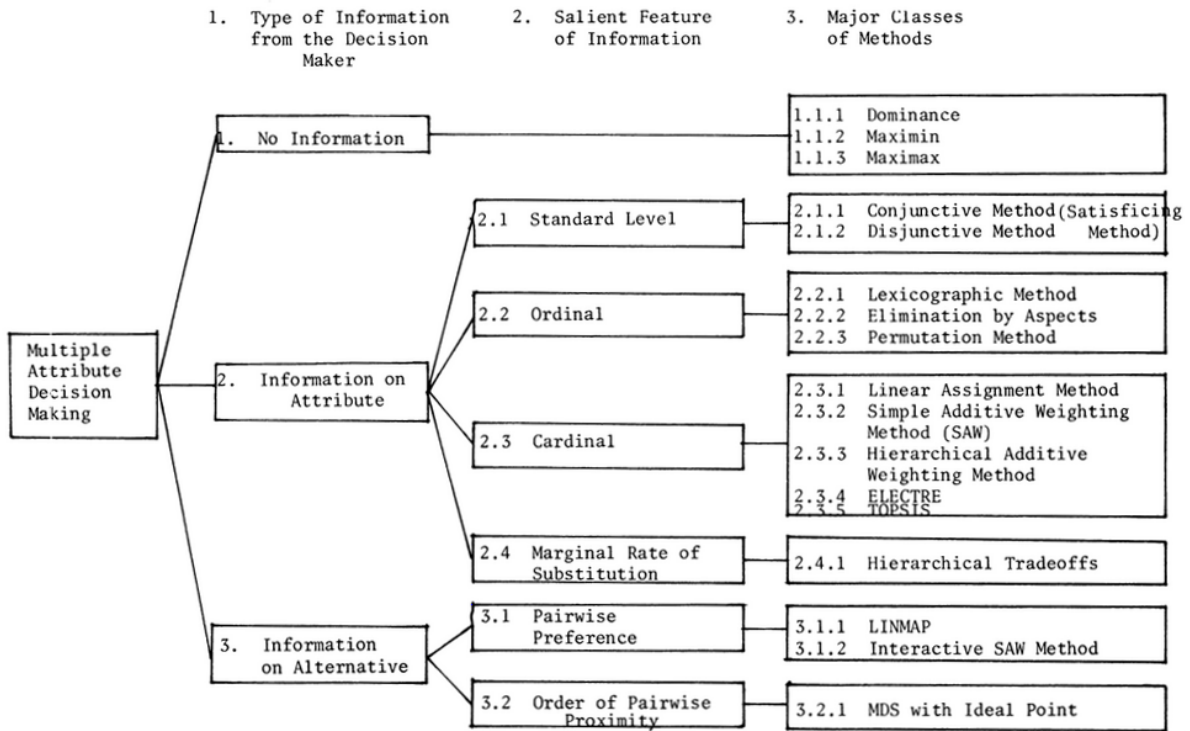


Figure 3.1: Classification of MADM methods (Source: [Hwa81]).

Furthermore, as different applications have different needs, the weights are highly subjective and offer flexibility. If a criterion is assigned a weight of zero, it should be omitted from the tree entirely since it will not affect the final ratings. Next, a decision matrix is created by rating each alternative regarding each leaf criterion (e.g., between 0 and 10). The final ratings for inner criteria can now be determined in a bottom-up manner by normalizing the weights of all sub-criteria for a given parent criterion and then establishing the parent’s composite rating as the sum of all sub-criteria ratings multiplied by their normalized weights. The ratings calculated at the root of the target tree are the final values of benefit of the alternatives. Accordingly, the highest-rated alternative is the best suited one.

Nonetheless, despite its simplicity and wide acceptance, the SAW method may not always be the best approach in its bare form. For example, it may be desirable for all criteria to achieve a minimal rating. In these cases, so-called conjunctive methods can be used [Fül05]. Disjunctive methods, on the other hand, need at least one criterion to meet a threshold for the alternative to be considered. These screening rules could also be combined with other approaches [Fül05]. Another approach can be found in the maximax or maximin methods which select the best alternative based on their highest or lowest rating, respectively. Additionally, the lexicographic method ranks all criteria based on their importance, and the alternative with the highest rating for the most important criterion is chosen. In case of a tie, the next important criterion is considered [Fül05].

In addition to these elementary methods, which are most useful for simple decisions involving few criteria, alternatives, and decision makers, various frameworks are available for more complex use cases. The analytic hierarchy process (AHP), for instance, involves creating a comparison matrix

by pair-wise comparing all criteria. Moreover, using varying methods, a set of criteria weights can be derived from the matrix which is as consistent with the comparisons as possible [Rao08]. In a similar manner, the individual ratings for subjective criteria can be determined by contrasting the performances of the alternatives. The final rating is ascertained using an aggregation method akin to other methods.

3.2 Assistance Systems

As a training tool, EXTRA is closely related to assistance systems, such as SCENAS and MOBAS (see section 2.1 on page 10), which have also been designed to train operators of complex military systems. The following section describes assistance systems in general and explains their relationship to EXTRA.

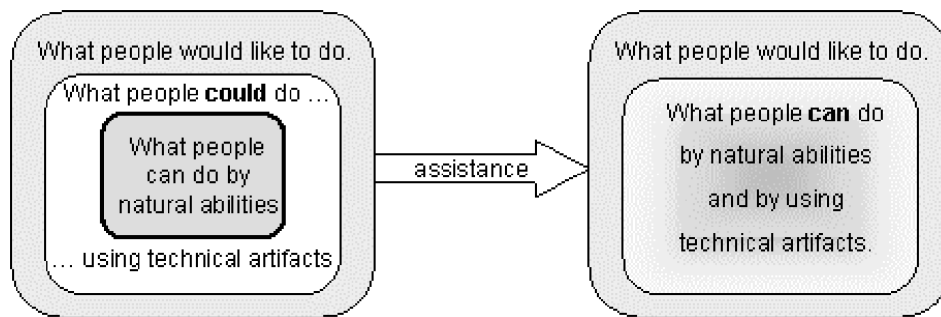


Figure 3.2: Assistance systems extend the capabilities of humans by allowing them to make full use of technological potential (Source: [Wan05]).

In the broadest sense, every technical artifact, including chain saws and cars, could be considered an assistance system [Wan05]. Wandke (2005) provides a narrower definition, as he describes assistance systems as man-machine systems that help users in performing tasks by giving them access to machine functions, and, thus, bridging the gap between technological potential and human capabilities [Wan05] (see fig. 3.2). In addition, unlike fully automated systems, they leave some functions to be performed by the user, while performing some on their own.

Consequently, assistance systems are often found when human operators are confronted with complex systems. Driver assistance systems such as braking and navigation assistants are popular examples [Naa94]. Other application domains include smart homes [Hei02], manufacturing [Kor15], and assistive technology for elderly and handicapped people [Kle07]. Assistance systems are also prevalent in the context of human-computer interfaces; for example, online help systems are used to make the interfaces more accessible [Elk87]. Moreover, in software engineering, intelligent assistance systems are used for automating repetitive tasks, finding code smells, generating code and documentation [Rec06].

In literature, various classification models applicable to assistance systems have been proposed. Sheridan (1988), for instance, classifies human-machine systems depending on their degree of automation in terms of decision support ranging from (0) *no assistance* through (5) *computer selects action and executes it if human confirms* to (10) *computer performs the job automatically* [She88]. Although this model offers a fine graduation, it is often too limited for assistance systems, which becomes evident when attempting to classify braking assistants [Wan05].

Wandke (2005) offers a more general taxonomy, where the decision-making process and the selection and execution of actions represent only two out of six action stages [Wan05]. Each of these stages involves different types of assistance. To illustrate, during the motivation stage, the system may provide coaching or warning assistance to either reinforce or diminish user intentions or it may provide advisory or delegation assistance during the decision process. Considering that most assistance systems support multiple types of assistance, each of which may be more or less appropriate in a given situation, an additional adjustment dimension is included. This dimension encompasses the following four approaches: *fixed* (no adjustments), *customization* (adjustments during design), *adaptability* (adjustments by the user), and *adaptation* (adjustments by the system).

In Wandke's taxonomy, the last action stage is characterized by providing feedback, which comprises general feedback about the state of the system (e.g., LED turns on after device is powered on) and the evaluation of user actions (i.e., critique and suggestions) [Wan05]. As such, especially critique assistance is closely related to learning and training systems and is most useful in scenarios where human operators are not substituted entirely by machines and his or her performance matters. Furthermore, Wandke draws an analogy between these types of assistance systems and teachers and trainers as they share the responsibility of providing feedback [Wan05].

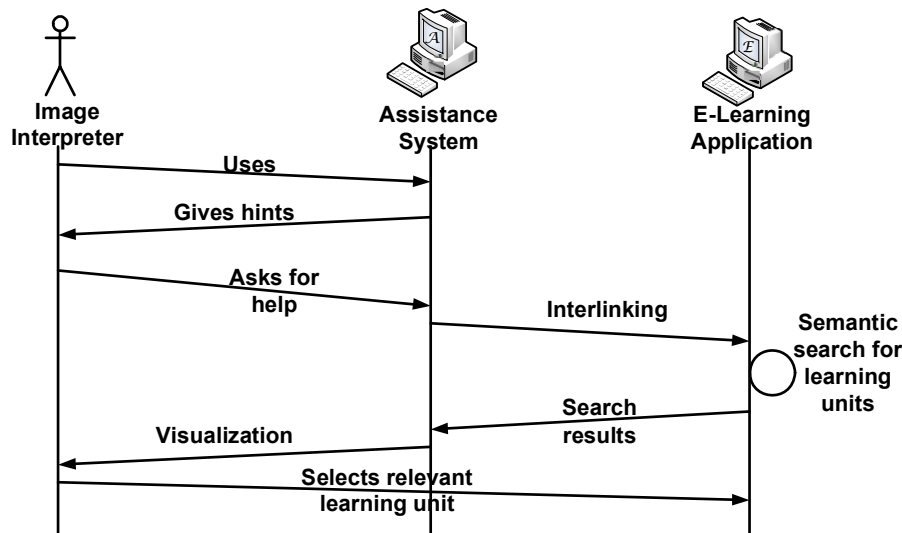


Figure 3.3: Interaction between user (image interpreter), assistance system, and e-learning application to automatically select learning units depending on the context (Source: [Str11]).

Another approach to combining assistance and learning systems can be observed in the intelligent interlinking of two such systems. For instance, in complex working environments (e.g., that of a radar image interpreter), assistance systems facilitate the workers' jobs, while e-learning system provide help and learning content [Str11]. Here, the linkage enables assistance systems to be enriched with the context-dependent provision of learning units. Instead of having users seek proper learning content themselves, the joint system can automatically search for content based on the current working context and the user model [Str11]. This interaction process is illustrated in fig. 3.3.

Ideally, an assistance system should be engaging and motivating to raise adoption and productivity. A popular approach to solving this problem is *gamification*, which can be defined as "the use of game-design elements in non-game contexts" [Det11]. Korn (2012) adopted this approach to make the otherwise dull and repetitive production work more appealing to elderly and disabled

individuals [Kor12]. Therefore, instead of having a solely function-oriented user interface, he proposed to visualize the assembly process in real-time in a fashion similar to Tetris bricks, while also giving (visually appealing) feedback about the work performance. Similarly, Magaña, and Organero (2014) applied gamification to an eco-driving assistant by scoring the drivers' performances, introducing achievements and allowing them to share and compare their results with friends and other users [Mag14]. Finally, Streicher and Szentes (2013) augmented their assistance and learning system called Mobile Assistant (MOBAS) with short quizzes to support and reinforce the learning process [Str13].

3.3 Digital Game-Based Learning

Prensky [Pre01a] coined the term Digital Game-Based Learning (DGBL) and this term is used as the conceptual basis for EXTRA. As Prensky explains, “digital game-based learning is precisely about fun and engagement, and the coming together of and serious learning and interactive entertainment into a newly emerging and highly exciting medium - digital learning games” [Pre01a]. It is motivated by the perception of a mismatch between today's educational system and today's students, which he refers to as “Digital Natives” as opposed to “Digital Immigrants” [Pre01b].

The concept of DGBL relates to serious games, which, generally, are “games that do not have entertainment, enjoyment, or fun as their primary purpose” [Mic05]. In contrast to digital learning games, serious games are not necessarily digital and are also utilized in fields other than education such as healthcare, military, corporate training, and advertising [Dja11]. As a result, serious games can be considered a superset of DGBL. Nonetheless, as any video game could be used for serious purposes, it makes sense to only consider those which have been designed for non-entertainment purposes right from the start [Alv08]. Figure 3.4 displays a more comprehensive classification of DGBL, serious games, and related concepts.

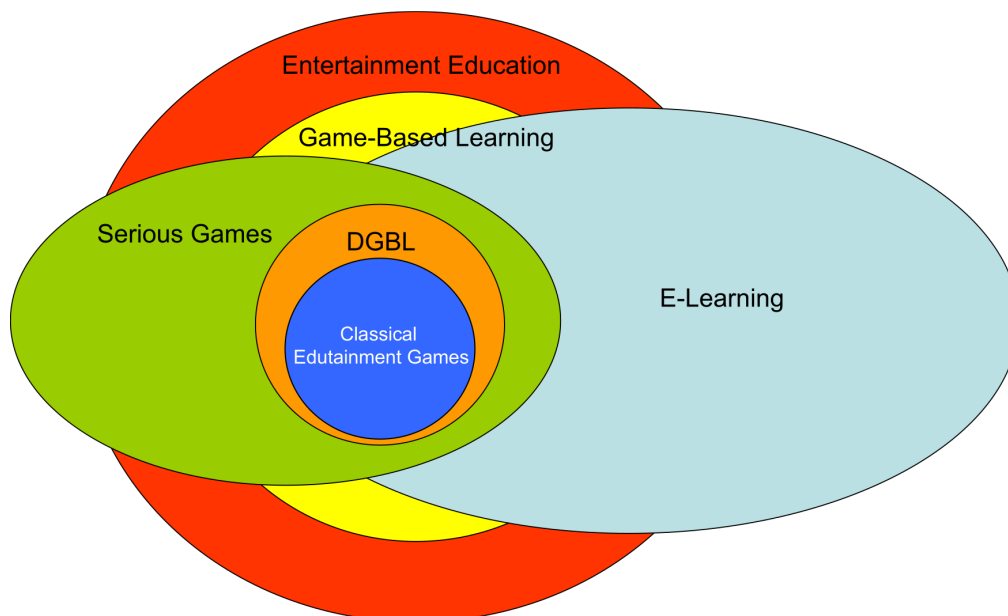


Figure 3.4: Possible classification of DGBL and related concepts (Source: [Bre10]).

Above all, the use of games for education is driven by the notion that motivation is a necessity for successful learning and that traditional educational methods are often considered “dry” or “technical”, i.e., “boring” [Pre01a]. On the other hand, motivation and engagement has always been the expertise of the video games industry [Pre03]. Moreover, educational games can contextualize learning materials to add engagement and interaction to the learning process.

Having established motivation as a key factor in both learning and games, the following paragraphs describe two psychological models that aid in understanding human behavior related to this phenomenon and which are applicable to games.

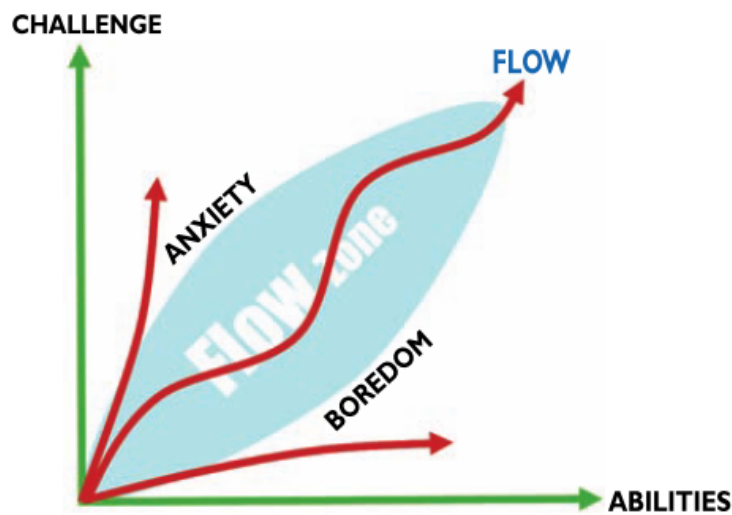


Figure 3.5: Flow zone: boundary between boredom and anxiety (Source: [Che07]).

One such model is the flow. Gamers often refer to the flow as (being in) “the zone” [Che07], and Csikszentmihalyi describes flow as the psychological “state in which people are so involved in an activity that nothing else seems to matter; the experience itself is so enjoyable that people will do it even at great cost, for the sheer sake of doing it” [Csi91]. In games, the difficulty should be on par with the players’ abilities to keep them in their individual flow zones (i.e., neither bore them nor cause anxiety; see fig. 3.5) [Che07]. Games can also ensure that the flow is never interrupted by offering adaptive choices and by embedding them into the core activities [Che07].

Additionally, B. J. Fogg designed another psychological model called *Fogg’s Behavior Model* [Fog09] (see fig. 3.6 on the next page). This model explains human behavior by the presence of the following factors: motivation, ability, and trigger. Motivation arises from three two-sided core motivators, which are pleasure and pain, hope and fear, and social acceptance and rejection, while ability revolves around the availability of resources such as time, money, and effort (simplicity factors). Finally, triggers cause people to perform a behavior at a specific moment. These triggers can be sparks (motivational), facilitators (lower ability requirements), or signals (indicators or reminders) [Fog09]. To apply this model to e-learning systems, Muntean (2011) concluded that, for learning to be effective, users must be at the same time motivated, capable, and triggered to reach the state of flow [Mun11].

Despite the apparent benefits of serious games as learning tools, there is still skepticism, especially from non-gamers [Che07]. The following section presents several aspects where existing educational games often prove inadequate and potential may remain untapped.

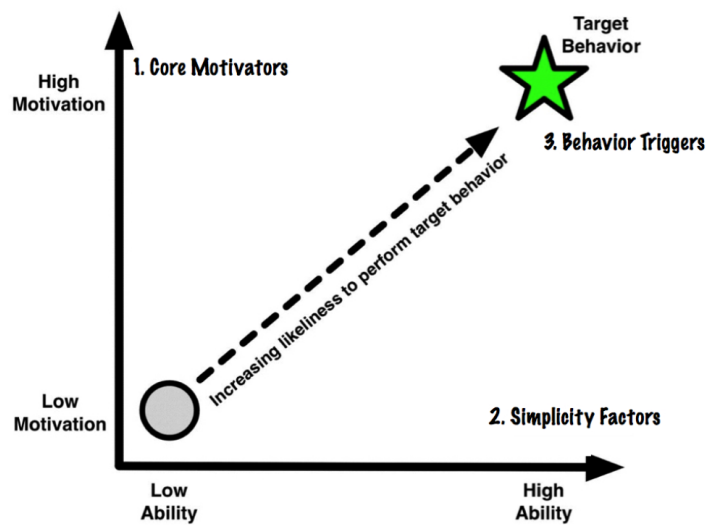


Figure 3.6: Fogg Behavior Model: High ability and motivation increase the likelihood for triggers to cause a specific behavior (Adapted from [Fog09]).

Combining game and didactic elements Naively embedding educational material in a game does not lead to an effective learning experience; instead, it is required to discover the optimal balance between entertainment and learning [Bre10]. Solutions that fall short in this regard are often called “Shavian reversals”, i.e., learning games that are neither enjoyable as a game nor effective as a learning tool [VE06].

Adaptivity and personalization Existing educational games are often targeted at a rough population of users, whereas they should rather focus on the individual user’s needs [Mag09, Str15]. For example, they can use automatic educational reasoning (AER) systems [Swe13b], such as intelligent tutoring systems (IRS), to oversee the learner, offer guidance, and select content, just like a human tutor would [Mag09].

Integration and authoring tools Swertz et al. (2013) argue that computers cannot play or teach, yet they can be employed to “create a playground for the game called teaching and learning” [Swe13a]. However, educational games are often standalone, requiring the game company for modifications [Mag09]. Supporting user-created content, authoring tools, and modding would make integrations into classrooms and other forms of education easier by putting authorship in the hands of the students and educators [Mag09].

3.4 HTML5 Game Engines

To avoid “reinventing the wheel” and to reduce development time, EXTRA utilizes the HTML5 game engine Phaser (phaser.io), which was deemed most suitable in chapter 4 on page 31. This section describes the ideas and considerations related to the use of (HTML5) game engines.

Game engines supply the foundation for many different games without major modification [Gre09]. They are usually organized in a layered architecture, with their runtime components building upon hardware, drivers, and operating systems. Game engines are made of a variety of components

or subsystems that handle responsibilities such as graphics, audio, user input, physics, collision detection, and networking [Gre09]. In addition to the runtime component, they often come with tool suites to ease the creation of additional content such as textures and animations [Gre09].

Early games, such as PacMan, were created without game engines and were designed to only work on one specific type of hardware [Gre09]. Later on, first-person shooter games, such as Doom, introduced a clear separation between core software components and game-specific assets, enabling developers to customize various aspects of the game and thus giving birth to the modding community. In response, later games (e.g., Quake III Arena) were built with reuse and modding in mind [Gre09]. Due to this development, the line between a game and its engine is often blurry; some games clearly differentiate between the two, while others do not attempt to do so [Gre09].

Ideally, a game engine would support creating virtually any type of game. However, such a general-purpose engine is not realistic. As usual in (software) design, trade-offs need to be considered. Here, the trade-off concerns re-usability and optimality; hence, the more general-purpose game engine is, the less optimal it is for a particular game and target platform [Gre09]. As a result, many game engines are only suitable for a specific game genre.

In contrast to conventional engines, HTML5 game engines build upon web technologies instead of low-level hardware and operating systems. Therefore, they are often more limited in their scope as they do not need to abstract low-level APIs or provide for cross-platform support themselves. For example, the web audio API can be used to play and control a wide range of audio formats on various platforms, while the canvas API allows drawing primitives such as basic shapes, texts, lines, and images.

In addition, web technologies provide several, convenient APIs for networking; AJAX can be used to realize asynchronous HTTP request using the XMLHttpRequest API, and the more recent WebSocket API and protocol allows for bidirectional connections between clients and the server. Finally, HTML, CSS, and the Document Object Model (DOM) API provide a way to create complex, dynamic layouts and handle user input. As many websites rely on these technologies, they clearly have proven themselves in practice. Figure 3.7 illustrates two different games realized with Phaser, and an overview of all HTML5 game engines used for the evaluation can be found in section 4.1 on page 32.

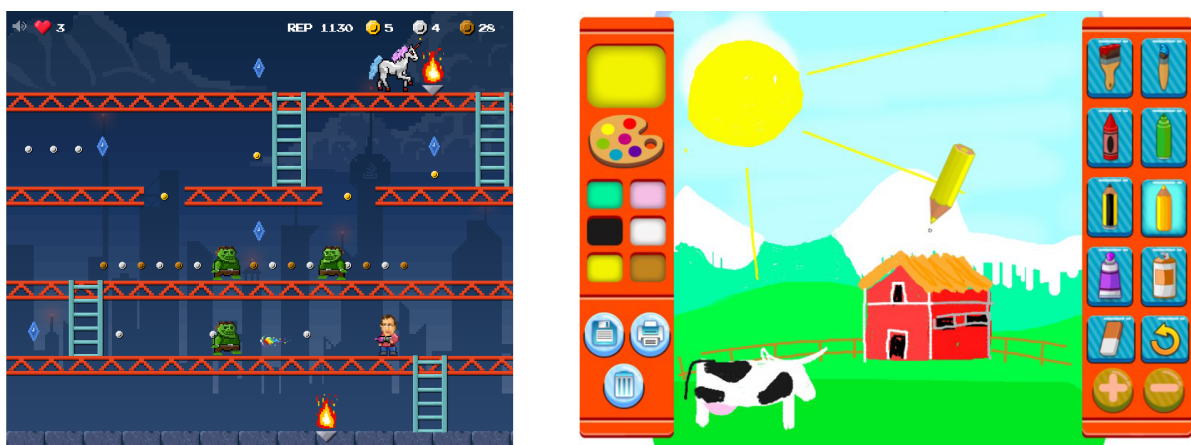


Figure 3.7: Two games featured on phaser.io: A platformer game called *Unikong* [Sta16] (left), and *Paint Online* [Tor16], a painting game for children (right). Source: [Pho16].

3.5 Web Development

The web is an attractive application and game platform for several reasons [Tai11]. First, applications can be built to run on a variety of devices as long as they can run a web browser. Second, these applications can be started by simply opening a URL in the browser. Furthermore, web applications can usually be deployed and updated with relative ease and do not require any specialists or high-cost hardware. In recent years, NodeJS enabled web developers to run JavaScript (JS) programs without the need for a browser, making it possible to use JavaScript both server-side and client-side and even to share code between these applications.

As applications are becoming increasingly complex, it is desirable to have a sophisticated toolset to manage and ease development, testing, and deployment. Fortunately, such tools can be written in JavaScript using NodeJS, which is why the repository of NodeJS's package management system features a large set of them.

The following section commences with an overview of JavaScript, followed by descriptions of the tools used for EXTRA, including the motivation for using them.

3.5.1 JavaScript

JavaScript is the only programming language, which natively runs in web browsers, that does not require additional plugins unlike technologies such as Flash and Java applets. To not require the installation of such plugins, EXTRA is entirely programmed in JavaScript.

In 1995, Brendan Eich originally designed JavaScript within a mere 10 days [Sev12] and the programming language has evolved rapidly since then. As a multi-paradigm language, JavaScript is often misunderstood and despised because developers expect quick results and wrongly blame JavaScript for browser inconsistencies and unwieldy browser APIs such as the DOM API [Cro08]. Yet, as Crockford (2008) highlights in his work *JavaScript: The Good Parts* [Cro08], JavaScript can be a productive programming language, provided the right tools and language features are used.

The implementation of JavaScript is specified by the ECMAScript standard of the European Computer Manufacturers Association (ECMA International) as ISO/IEC 16262 and ECMA-262 [ECM15]. JavaScript's latest iteration is called ECMAScript 2015 (ES2015), after being renamed from ES6. Nonetheless, even in modern browsers, many of its new features have yet to be implemented. Fortunately, various tools exist to enable the translation from ES2015 to the older and more browser-compliant ES5 standard from 2009.

By utilizing such tools, EXTRA uses several new features added in ES2015 (e.g., the module system and the class syntax) that help in terms of code organization and expressiveness. Listing 3.1 highlights a number of the relevant additions, and a more comprehensive list can be found on, for example, es6-features.org.

```
1 // constants
2 const MY_CONSTANT = 1;
3 MY_CONSTANT = 2; // error
4
5 // arrow functions
6 odds = evens.map(v => v + 1);
7 pairs = events.map(v => ({ even: v, odd: v + 1}));
8
```

```
9 // modules
10 import { sum, PI } from './lib/math';
11 export function addPi(n) { return sum(n, PI) }
12
13 // classes
14 class Circle extends Shape {
15     constructor(x, y, radius) {
16         super(x, y);
17         this.radius = radius;
18     }
19
20     getDiameter() {
21         return this.radius * 2;
22     }
23 }
24
25 const myCircle = new Circle(1, 2, 4);
26 console.log(myCircle.getDiameter()); // prints 8
```

Listing 3.1: Some of the new language features in ES2015 (Source: es6-features.org).

3.5.2 NodeJS

The NodeJS JavaScript runtime environment is used to run the server component of EXTRA and for the development of the game client (see section 5.4 on page 47). NodeJS is built on Chrome’s V8 JavaScript engine and employs an “event-driven, non-blocking I/O model that makes it lightweight and efficient” [Nod16]. Now in the hands of the Node.js Foundation, it was originally created by Ryan Dahl and publicly announced in 2009 [Dah09]. His motivation was to write HTTP servers in a simple manner. Realizing that no existing solution suited his needs and that “I/O needs to be done differently” (i.e., based on non-blocking I/O instead of multiple OS threads), he turned to JavaScript due to its event-based nature [Dah09].

Although NodeJS was originally designed for server applications, it is currently also used in other ways (e.g., as a development platform for client-side applications), which may be due to its powerful command-line integration, its cross-platform support, the package manager that comes with it [McA14], and because it uses JavaScript. In addition to being familiar to many developers as the only programming language that natively runs in the browser, JavaScript also provides other practical benefits; for instance, it makes sharing code between server-side and client-side applications easier and is interpreted very fast nowadays due to browser competition [Spr13].

An important aspect of NodeJS is the *Node Package Manager* (npm). Measured in pure module counts, npm features the largest ecosystem of open-source libraries [DeB16], including server-side frameworks, client-side tools (e.g., for workflow automation), and general-purpose libraries.

Another part of NodeJS is the *CommonJS* module pattern, which allows modules (files) to import other modules using a function called `require` and to export any object by assigning it to `module.exports`. Next to providing structure, this approach to modularization also eliminates the need for a globally shared scope, which would otherwise cause variable names to conflict between different modules. Unfortunately, there is no comparable module system that runs natively in the browser today.

3.5.3 Webpack

EXTRA uses Webpack (webpack.github.io) to automate the build and bundling process of both the client and server component in a flexible way. Webpack describes itself as a JavaScript module bundler that processes different modules including their dependencies and generates static assets representing those modules as depicted in fig. 3.8 [Kop16]. Tobias Koppers created Webpack, and it is an open-source project written in JavaScript.

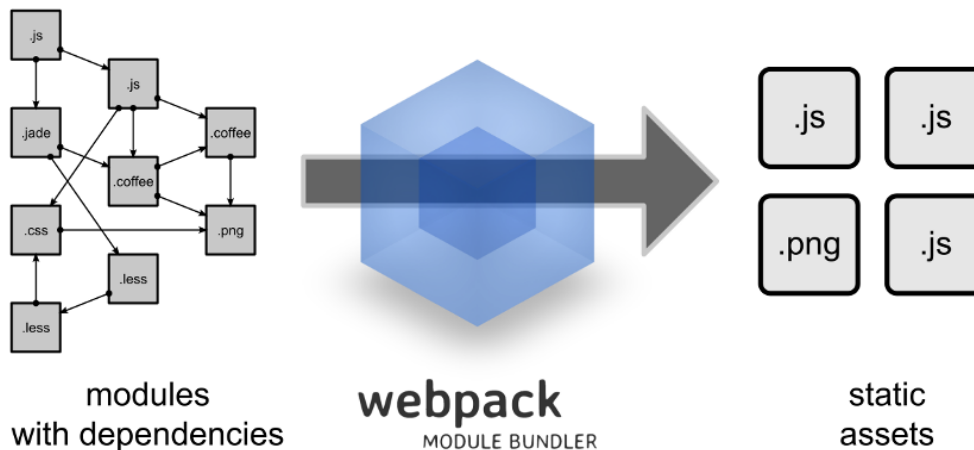


Figure 3.8: The overall bundling process of Webpack (Source: [Kop16]).

Furthermore, using a module bundler, such as Webpack, provides several benefits for modular client-side web applications, especially in terms of maintenance efforts. In the simplest scenario without a bundler, developers would manually maintain a list of script tags, while the scripts would have to communicate through the global scope. Not only can this lead to hard-to-trace bugs caused by the shared global scope, but the implicit dependencies also make it easy to arrange the script tags in the wrong order, accidentally leave out dependencies of present scripts, or to forget to remove unused scripts. Moreover, there is a one-to-one correspondence between the modularization chosen by the developer and the files (modules) downloaded by the browser.

Module bundlers such as Webpack, on the other hand, allow the dependencies to be specified explicitly inside the consuming modules similar to many other programming languages; thus, enabling dependencies to be traced automatically. As such, users can define one or more entry scripts which are subsequently bundled together with their dependencies into a single output script that can be executed in a browser. In order to accomplish this, Webpack needs to statically analyze the code. Various module formats to specify dependencies are supported out of the box, the most important being CommonJS also used by NodeJS.

One of the core features of Webpack are loaders. By default, Webpack supports only normal JavaScript modules written in plain JavaScript (ES5). Loaders, however, allow various file formats, such as images and style sheets, to be integrated in a meaningful way. To illustrate, referenced style sheets can be wrapped in a JavaScript module to automatically apply them during runtime, or they can be concatenated into a single CSS file emitted alongside the final JavaScript bundle.

3.5.4 React

As a result of the evaluation presented in chapter 4 on page 31, React was chosen to realize the user interface (UI) of EXTRA. React, made by Facebook and first released in 2013, allows to build UIs by creating and composing multiple components [Hun13]. Its goal is to solve challenges that accompany developing complex UIs involving data that changes over time [Gac15]. Furthermore, React is often wrongly compared to MV* frameworks (e.g., MVC, MVVM, or MVP [Osm12]), such as Backbone, Ember.js, and AngularJS, when, in fact, it focuses just on the view, making it easy to integrate into existing technology stacks.

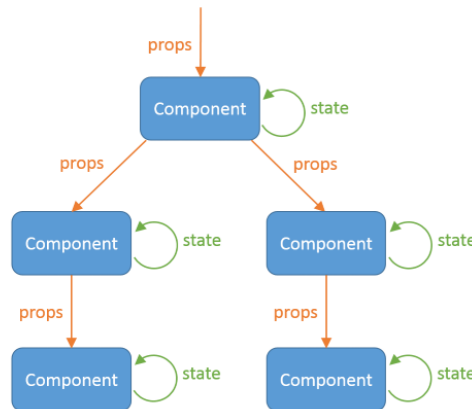


Figure 3.9: Component model and data flow in React (Source: koalite.com).

At the very core of React are components (e.g., in the form of classes) that can be composed in a hierarchy as illustrated in fig. 3.9. Each component can define several lifecycle methods and must define a `render` method that declaratively describes its rendering output (a tree of other components and DOM nodes) at any given point in time. Ideally, the `render` method is a pure function of the component's props and state, i.e., it has no observable side effects and always returns the same result for the same props and state. Props in React refer to properties passed down from the parent component in its `render` method, while state denotes the internal state of the components. Similar to HTML attributes, props are also commonly used as a way to pass callbacks to child components, e.g., click handlers.

Listing 3.2 exhibits a simple Hello World example in React using ES2015 and the JavaScript XML (JSX) syntax extension, which was invented by Facebook to increase readability and can easily be translated to plain JavaScript.

```

1 // component definition
2 class Hello extends React.Component {
3   render() {
4     return <div>Hello, {this.props.name}</div>;
5   }
6 }
7
8 // rendering the component into a DOM element
9 React.render(<Hello name="World" />, document.body);

```

Listing 3.2: Hello World example in React using JavaScript XML (JSX).

Whenever a component changes its state or is given new properties, it needs to re-render itself and all its descending components, which results in a representation of the desired DOM state. Since DOM operations are expensive, React maintains an abstract copy called virtual DOM. This way, React can determine the minimal set of DOM operations using diffing algorithms, thus providing a bridge between the imperative DOM API and the declarative React component specification.

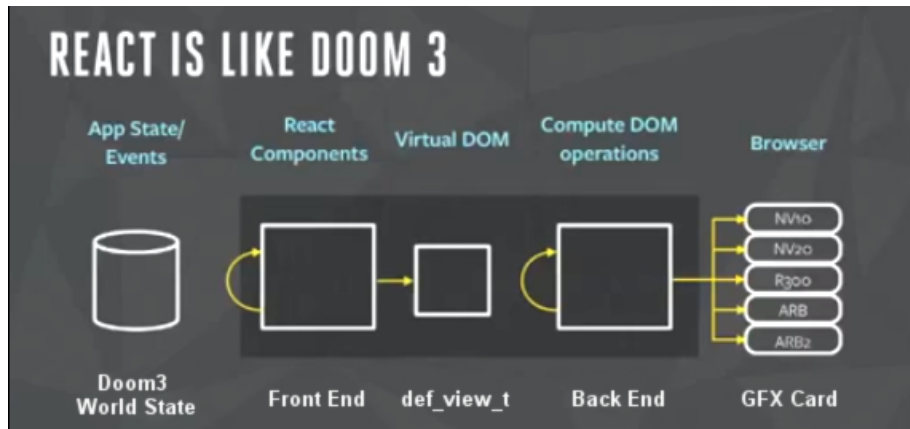


Figure 3.10: Comparison of React and the rendering engine of Doom 3 (Source: [Hun14]).

Pete Hunt, a developer who used to work on React at Facebook, compared this approach to the rendering engine of Doom 3, which, similar to React, translates an intermediate scene representation based on the world state into a set of low-level (OpenGL) operations (see fig. 3.10) [Hun14].

4

Game Engine Analysis

The purpose of the following analysis is to systematically determine the engine best suited for the implementation of EXTRA. As explained in section 3.1 on page 17, the method is based on the proposal of Szentes, Bargel, and Streicher [Sze11]. The essential steps and their results are:

1. Selection of alternatives (*list of alternatives*)
2. Selection of criteria (*criteria tree*)
3. Weighting of criteria (*weighted criteria tree*)
4. Evaluation of each candidate against the leaf criteria (*decision matrix*)
5. Calculation of composite and final scores (*ranking of alternatives*)

During the analysis, it turned out that none of the considered game engines included a system to create sophisticated user interfaces (UIs) on top of the game world (i.e., heads-up displays, or HUDs) that goes beyond the absolute positioning of simple texts, images and primitives. As an economic simulation game that may include online features (e.g., leaderboards) in the future, however, EXTRA requires the use of context-sensitive panels and complex layouts. Therefore, two popular HTML5 UI libraries were included and their ratings were systematically combined with those of the game engines to yield a final rating for the combined solution.

Even though this combined-solution approach inherently increased the complexity of the overall solution, the reliance on HTML and CSS seems like a reasonable approach, considering that more and more games with heavy UIs are doing the same, including AAA games on platforms other than the web [Nik15]. For example, Electronic Arts maintains an open-source fork of the WebKit rendering engine, which they use for SimCity (2013), Skate 3, Sims, and even for console games [Chi15].

Besides the advantage of being able to draw on existing technology and tooling, HTML and CSS should make it easier for other developers to work on the UI of the project, since many developers are already familiar with web technologies and because many resources are available on the internet. In addition, this approach allows for the UI to be decoupled from the game engine itself, reducing potential switching costs in the future.

To integrate the combinations of game engines and UI libraries into the analysis, an appropriate aggregation method was assigned to each leaf criterion. For example, `max` was used for features that are only required as part of either the engine or the UI library, while `avg` was mostly used for criteria related to documentation and stability, as these factors are desirable in both products. Finally, `min` was used for licensing and cost criteria, because they represent a limitation to the combined solution.

This chapter is subdivided as follows: section 4.1 provides an overview of the considered game engines and UI libraries, while section 4.2 on the next page explains the selected criteria and their weights. The final results are presented in section 4.3 on page 34 and discussed in section 4.4 on page 36.

4.1 Overview of Alternatives

Five candidate 2D engines were selected from html5gameengine.com based on their rating, popularity and last release date. Full-fledged “game makers” such as Construct 2 that come with their own development environment to allow the creation of games without writing any code were not considered for reasons of flexibility and to make sure that other JavaScript libraries can be easily integrated.

Due to time and resource constraints, only two UI libraries were included, namely React and Ember.js. They were selected from GitHub’s showcase of frontend frameworks. AngularJS was another consideration, but was not included because it was undergoing a radical transition from AngularJS 1 to AngularJS 2 at the time of the analysis, the latter of which was still in development (beta version).

The following sections summarize all selected game engines and UI libraries.

Web-Based Game Engines

Pixi.js Although more of a 2D (canvas) rendering engine than a complete game engine, Pixi.js (pixijs.com) provides a fast and lightweight solution for cross-platform applications and games with rich and interactive graphics. It does not support features like top-level state management, physics and audio, however. Besides traditional canvas rendering, Pixi.js also seamlessly supports WebGL rendering for better performance [Goo13].

Phaser With three included physics systems, Phaser (phaser.io) is arguably the most feature-packed of all considered game engines. It is described as a “fast, fun and free open source HTML5 game framework” [Pho16]. Nonetheless, an adequate amount of learning resources can be found on the official website. It is built upon Pixi.js, although this is subject to change in the next major release [Bar15].

melonJS Another feature-rich game engine can be found in melonJS (melonjs.org). It focuses on performance and has built-in support for the *Tiled Map Editor*¹ and the map exchange format *Tiled Map XML* (TMX) allowing for an easy design and exchange of game levels in an interoperable way.

CreateJS Similar to Pixi.js, CreateJS (createjs.com) was not primarily made for game development, but for building rich and interactive content on the web. It is a suite of four different libraries designed to work together, the most important being EaselJS which is responsible for rendering to a canvas. The other libraries are SoundJS, PreloadJS, and TweenJS, which can be used for playing audio, preloading assets, and performing animations, respectively.

¹ Tiled Map Editor is an open-source WYSIWYG editor that allows the design of levels by loading tileset images, placing the contained tiles on a grid, and assigning custom attributes to them. See <http://www.mapeditor.org/>.

Crafty Unlike the other considered game engines, Crafty (craftyjs.com) uses an Entity-Component-System architecture, i.e., all entities (game objects) are assigned a set of components that specify their behaviors (e.g., `Canvas` for rendering, `Gravity` for physics, and `Draggable` to enable drag and drop), while systems are used for other objects that handle entities and make use of the event system. Crafty's feature set is similar to those of Phaser and melonJS.

User Interface Libraries

Ember.js Ember.js (emberjs.com) is a comprehensive and component-oriented framework for creating ambitious web applications [Til16]. It focuses on productivity and developer ergonomics, and integrates the popular *Handlebars*¹ templating engine to describe the views of each component, which are automatically updated by the framework whenever the underlying data changes.

React In contrast to Ember.js, React (reactjs.com) is only concerned with the view of the application and hence does not describe itself as a framework. As such, it does not include a routing API and does not have a notion of controllers or services. Similar to Ember.js, however, it uses components as the building blocks for UIs. A more detailed description of React is given in section 3.5.4 on page 28.

4.2 Criteria Tree

The criteria tree created for this analysis can be seen in fig. 4.1 on the next page. It is based on the one suggested in [Sze11], but was adapted by adding and removing criteria to reflect the requirements of EXTRA in a better way.

Each criterion was given a weight between 1 and 10 according to its importance to the project based on requirements derived from the game design concept and the requirements that seem likely to arise in the future.

License This category comprises license costs and type, where license type is a special type of criteria classified as *fatal*, i.e., candidates can only score 0 or 10 points, and a score of 0 means that the candidate is discarded regardless of its other ratings. In this case, the engine or library must not prohibit commercial use or be infective (copyleft).

Documentation A good documentation and quality resources make it easier for new developers to work on the project. Therefore, this category is weighted relatively high with 6 out of 10 points. The category consists of API specification, tutorials and examples.

Development Factors such as the availability of a component or plugin ecosystem, debugging utilities, and community support are useful for a productive development process, while supporting a wide range of platforms makes the final game more accessible.

¹ Handlebars allows writing templates as HTML snippets that can contain variables and control structures using a domain-specific language. These templates can then be combined with data models to produce HTML code. See <http://handlebarsjs.com/>.

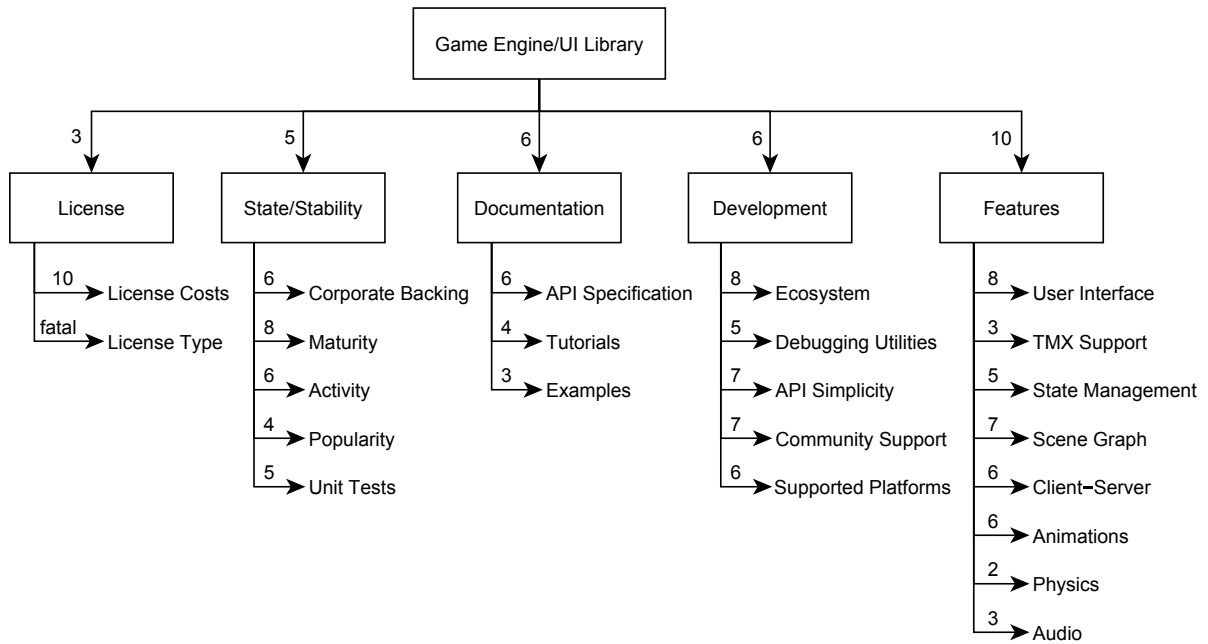


Figure 4.1: Criteria categories, subcategories and their weights.

State/Stability A mature and stable game engine or UI library should be more reliable and cover more essential features which may have been overlooked in this analysis. Popular engines which are still being actively developed are less likely to be abandoned soon and more likely to implement new features as the underlying technology advances. Criteria such as corporate backing, popularity, activity, and unit tests contribute towards the trust and longevity of the game engine.

Features Built-in support for the included features facilitates the development of the game and are the very reason to use a game engine or UI library. It comprises common features usually found in game engines such as (top-level) state management, scene graphs, and animations. Some criteria like audio are weighted lower than their apparent importance because browsers already provide a high-level and easy-to-use API, or because the expected effort to integrate another library to provide this particular feature is low.

4.3 Results

After creating the criteria tree, all candidates were evaluated against the leaf criteria using a score between 0 and 10. To make this process more transparent, the criteria were evaluated based on a schema that roughly describes the expected qualifications for a certain score range.

Afterwards, the composite scores by category and the final scores could be calculated based on these evaluations and the criteria weights. The final scores are shown in table 4.1 on the facing page. The individual scores and evaluation schemas can be found in appendix A on page 77.

For example, the criterion (plugin or component) ecosystem in the category development had a weight of 8 out of 10, and candidates were rated 0-3 for this criteria if they featured *no or only few*

Table 4.1: Results of the analysis: categorical and overall scores for each considered game engine, UI library and their combinations. The winning combination is highlighted in gray.

		License	Documentation	Development	State/Stability	Features	Total
Weight		3	6	6	5	10	
Game Engines	Pixi.js	10	5.2	4.5	8.0	3.6	5.5
	Phaser	10	7.6	7.2	7.4	7.6	7.7
	melonJS	10	4.2	5.1	7.1	6.8	6.3
	CreateJS	10	4.9	4.2	7.9	4.4	5.6
	Crafty	10	5.2	4.2	6.3	5.7	5.8
UI Libraries	React	10	7.4	9.2	10	3.4	7.1
	Ember.js	10	6.6	8.9	9.4	3.4	6.8
Combined	Pixi.js + React	10	6.3	6.0	9.0	5.0	6.6
	Pixi.js + Ember.js	10	5.9	5.7	8.7	5.0	6.4
	Phaser + React	10	7.5	7.1	8.7	7.9	8.0
	Phaser + Ember.js	10	7.1	6.9	8.4	7.9	7.8
	melonJS + React	10	5.8	6.2	8.6	7.5	7.3
	melonJS + Ember.js	10	5.4	5.9	8.3	7.5	7.1
	CreateJS + React	10	6.1	5.9	9.0	5.7	6.8
	CreateJS + Ember.js	10	5.8	5.5	8.7	5.7	6.6
	Crafty + React	10	6.3	5.7	8.2	6.7	7.0
	Crafty + Ember.js	10	5.9	5.3	7.8	6.7	6.8

plugins, 4-7 if there were *multiple plugins available* and the engine included a *dedicated API*, or 8-10 if there were *many actively maintained plugins* and a *web repository* for easy discovery.

To evaluate the leaf criterion ratings of the combined solutions, the individual ratings were combined in a reasonable way, using common aggregation methods such as *max*, *min* and *avg*, depending on whether the criterion was equally desired in both components (*avg*), in only component (*max*), or whether a single low rating presented a limitation to the final product (*min*).

As can be seen in table 4.1, the combination of the *Phaser game engine* and the *React UI library* scored highest and was therefore chosen for EXTRA. The second-best combination would have been Phaser and Ember.js, followed by the combinations involving melonJS.

The high overall rating of Phaser can be traced back to the categories documentation and features. In terms of documentation, Phaser excelled at providing a large set of tutorials and an exhaustive array of categorized examples, which run in the browser and can be edited directly on the web page. In contrast, melonJS and CreateJS only include few examples as part of their code repository. As for the features category, Phaser reached consistently high scores in most sub-criteria. It mainly stood

out at the physics criterion since it features three different physics systems. With a weight of 2 out of 10, however, this criterion contributed little to the final score. Phaser was closely followed by melonJS in this category.

The following sections summarize the results of each category.

License All engines and libraries achieved the maximum rating of 10. This is because all of them were open-source, free of cost and were either MIT or BSD (3-clause) licensed, both considered to be permissive and free.

Documentation In terms of documentation, which comprised API specification, tutorials and examples, Phaser and the two UI libraries scored highest. As mentioned before, Phaser's score was mostly due to the provided examples. The UI libraries, on the other hand, scored comparably low in this regard, but offered outstanding tutorials and API specifications. The low score of CreateJS and Pixi.js in this category was mostly a result of the unavailability of comprehensive tutorials and examples.

Development Phaser and both of the two UI libraries achieved high ratings in this category. All three reached the highest possible score for the criterion community support. Additionally, both UI libraries come with a comprehensive component ecosystem, as they are designed around building reusable components. The factor debugging utilities shows large gaps with Pixi.js and CreateJS providing none, while the two UI libraries even offer plugins that extend the browsers' built-in debugging facilities.

State/Stability All candidates reached a score of 10 for maturity, as they were all first released by 2014 and considered production-ready. With the exceptions of Crafty and CreateJS, the considered libraries and engines were also rated 10 for their recent activity, since they have been updated several times in the past year to fix bugs and include new features. React is the only candidate to score perfectly in all sub-criteria.

Features In this category, most game engines scored low in terms of user interface as none of them included sophisticated functionalities such as layout management to automatically position UI elements depending on the window size and the given layout hierarchy and parameters. This functionality is valuable to EXTRA since EXTRA requires many static elements and should support a wide range of devices with different screen sizes. Without this feature, the positions and sizes of the individual elements would need to be determined manually, which is a more tedious process.

Overall, Phaser and melonJS were found to adequately support many of the desired features. Pixi.js and CreateJS delivered relatively low scores, as they do not specialize in games. The same is true for the two UI libraries. None of the considered candidates come with built-in client-server support, i.e., they are all focused on the client-side, leaving potential server-side software to be developed and integrated by the user.

4.4 Discussion

The winning combination of the Phaser engine and React library later turned out to be a satisfying choice with decent support for most of the desired features and a strong community, making it easier to find solutions for common problems. The fact that Phaser scored high in both features

and community-related criteria could result from feedback effects, i.e., the more features there are, the more popular it becomes, which in turn results in more community feedback and engagement, leading to more and better documented features.

As already mentioned, however, the results of this analysis are subjective by design and are not applicable to other game development projects without further adjustments. To counteract these limitations, and to produce more objective and reproducible results, one could include stakeholders and domain experts at different stages. Still, a certain flexibility is desired to accommodate for different requirements.

Another limitation of this analysis is that it only includes two UI libraries. However, as complex web applications are becoming more commonplace, many UI libraries and frameworks are available for this purpose. The upcoming *Web Components*¹ standards by the W3C could have been a consideration as well, but were neglected since most of them are still not finalized (*Working Drafts*) [W3C16] and because there is some controversy among browser vendors [Har16].

The leaf criteria evaluations turned out to be difficult in many cases. For example, the criterion popularity was evaluated based on the amount of popularity stars on GitHub. Similar to the count of (Facebook) like button clicks, this metric may be biased, since people may star a project without actually using it, and since GitHub stars are never removed, even if the users switch to another library. Furthermore, stars might be lost after renaming the library or changing ownership, and stars are difficult to compare if the library is split up into several GitHub projects, as was the case for CreateJS.

Additionally, the evaluation of the category state and stability proved to be difficult as well, since the desire for a stable, yet actively developed game engine that includes many new features over time, is somewhat contradictory, and challenging to measure in isolation. Ideally, factors such as following a consistent and transparent versioning strategy, avoiding breaking changes, or at least providing clear upgrade guides, and the amount, severity, and time to fix bugs would have been considered, but were out of the scope of this analysis. Surveying the users of the game engines could have been fruitful as well.

The approach of including UI libraries and combining them with the game engines led to some questionable results, with the two UI libraries reaching a higher overall score than many of the game engines. This is mostly a result of the linearity of the SAW method and the limited contribution of the features category, allowing the two libraries to gain a high overall score throughout the remaining categories. A more adequate analysis method might treat certain factors as interdependent instead of isolating them, e.g., by limiting the score for documentation using the score for game-relevant features.

¹ Web Components comprise a set of upcoming standards that enable developers to create custom, reusable and encapsulated HTML elements using templates and imports [W3C16].

Exercise Trainer

This chapter describes the concepts and realization of the developed pre-exercise training tool called Exercise Trainer (EXTRA). The primary goal of EXTRA is to deliver a playful learning experience that engages and motivates a wide range of participants by exploiting the concepts of DGBL (see section 3.3 on page 21).

EXTRA is a web-based serious game and uses the Phaser game engine and the React UI library. According to the results of the subjective VBA presented in chapter 4 on page 31, this combination was deemed most fitting. Furthermore, EXTRA is realized as a NodeJS module comprising a client component (the game itself) and a comparatively small server component. Moreover, EXTRA utilizes modern web technologies which have been introduced in section 3.5 on page 25.

The realization of EXTRA involved several stages; first, the target audience and the learning objectives have been defined as described in section 5.1. Second, the game concept was created (section 5.2 on the next page). Finally, the game's architecture and design could be specified, as presented in section 5.3 on page 43 and section 5.4 on page 47, respectively. In addition, the application of EXTRA is outlined in Section 5.4 on page 47. Yet, both the learning objectives and the game concept were stipulated in advance of this thesis.

5.1 Learning Objectives

The overall goal of EXTRA is to allow participants of large-scale military exercises, which involve complex technical systems and processes, to better prepare themselves. This is accomplished by providing orientation and mediating knowledge (see fig. 5.1 on the next page for an exemplary process). To this end, the learning objectives have been specified in advance to the design of the game itself and in close cooperation with experts in NATO multi-national exercises, including participants and planners. This elicitation process was performed in advance of this thesis, however.

The learning objectives entail the technical and procedural knowledge transfer, both from a micro and macro perspective. These objectives can be summarized as follows:

1. To understand the overall scenario with its different roles, activities, and data flows.
2. To comprehend how one's own activities or systems are integrated into the overall system of systems, e.g., where data is coming from and where produced data is transmitted to.
3. To understand the effects of missing and defective data.

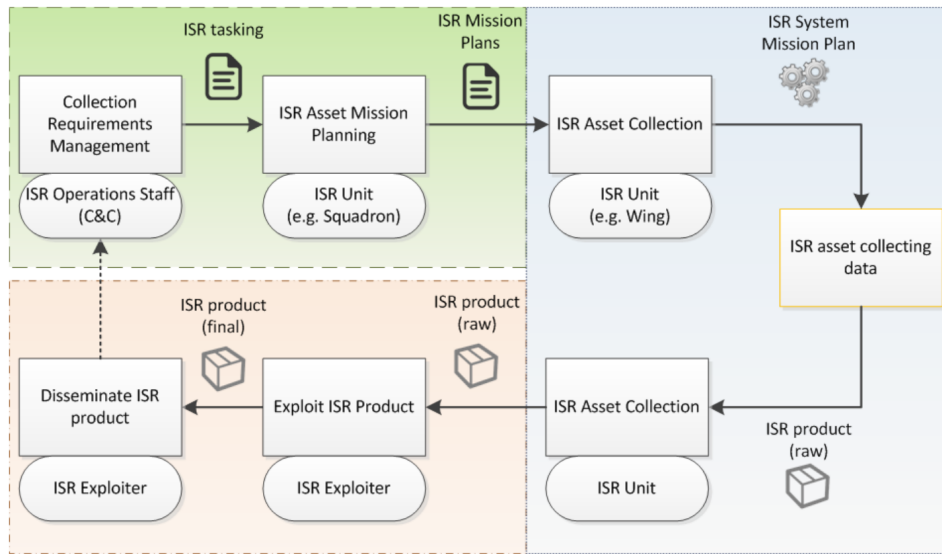


Figure 5.1: Example process for image acquisition in a joint exercise comprising input handling (top left), processing (right), and output handling (bottom left). Source: [Str16].

As the participants' initial motivation to prepare themselves is expected to be low, another goal was to be intuitively comprehensible and to not require a long time to become familiar with the game and its controls (high affordance). In addition, EXTRA should also intrinsically motivate the players by exploiting the concepts of digital game-based learning and game design in general. Furthermore, the game should be able to accommodate a wide range of heterogeneous users in different roles (e.g., soldiers, officers, and system operators from different nations), as is usually the case in large-scale exercises.

5.2 Game Concept

Similar to the specification of the learning objectives, the game concept was created in advance to this thesis at the Fraunhofer IOSB in the form of a Game Design Document. The concept is based on the learning objectives described in section 5.1 on page 39, which were used to determine the genre of the game (economic simulation). Furthermore, real-world elements have been carefully integrated to not break immersion and to increase acceptance.

The game features an isometric (2.5D) game world, in which the player has to serve customers by establishing production chains. Therefore, the player has to build, connect, and configure structures. Once a production chain is set up correctly, the end products are gathered and can be sold. In addition, the faster the customers are satisfied this way, the higher the player's score. Once no customer requests remain, the game is won. If the player takes too long, the customers will become annoyed and decrease the player's reputation, which represents his or her health and leads to losing the game if it reaches zero. In this game, the customers resemble requesting parties in the real world, and the structures stand for real systems.

An important design decision involved creating the game in a modular way so that the game elements and the scenario can easily be adapted to different exercises and changing requirements [Str16].



Figure 5.2: The modular game design concept of EXTRA (Source: [Str16]).

Additionally, since the game is built around logistics, the game could be applied to processes or systems other than those found in military scenarios [Str16]. This modularization approach is depicted in fig. 5.2 and can be described as the following distinct structural levels [Str16]:

1. *Technical level*: entails the modeling of the systems, processes, and data flows using standard formats, such as BPMN, UML, or SysML, for reasons of interoperability.
2. *Scenario level*: describes the actual scenario based on the technical level which may differ from exercise to exercise yet may not necessarily warrant a change to the technical level.
3. *Game level*: abstracts the components of the scenario level and enriches them with familiar game elements to deliver a playful experience, e.g., through narratives and game mechanics.

Figure 5.3 on the following page depicts the game's virtual world and user interface as realized in the prototype, while the following section describes the most important elements present in the game and their relationships to the real world:

Factory Factories represent roles, units, or data-processing systems. They are characterized by the processes or activities they support. While every process has a pre-defined output, some of them do not require any input (e.g., recording raw videos), while others do (e.g., video conversion). The production rate of each factory depends on the assigned workforce. Hence, it is up to the player to place, configure, and link the factories correctly in this regard, although they (and other structures) can also be pre-placed during the authoring of the scenario.

Market Markets are buildings that gather incoming products and add them to the inventory. Therefore, markets are always at the end of production chains. Depending on the market, different products are accepted. As such, markets correspond to target processes (e.g., dissemination) in the real world.

Route The game features three route types; streets, rails, and waterways, which differ in terms of speed, capacity, and their support for crossings. These routes are used to establish connections between factories or other buildings and resemble various communication channels, such as LAN or WAN, which vary in terms of bandwidth and connection quality.



Figure 5.3: Screenshot of the EXTRA prototype displaying the 2.5D game world and user interface.¹

Logistics Center Paraphrasing interface nodes (e.g., coalition shared databases) of the real world, the in-game purpose of logistic centers is to link two connections, thus enabling the use of different route types to connect factories and markets.

Connections The player can create directed connections in order to link two buildings (i.e., factories, markets, or logistic centers) and to realize the desired product or information flows. The player has to establish these connections as part of the gameplay and routes require to be present along their path. Additionally, the load of connections in terms of products per unit of time can be configured, yet is limited by the factories' production rates and the route capacity along the path.

Product Products resemble any kind of information that is produced in factories and flows between the various buildings. Each product has a distinct color for reasons of comprehensibility. Once collected in a market, they can be sold to the demanding customers.

Customer Customers represent parties whose demands must be met to win the game. For this reason, they can be considered opponents. Customers are characterized by the type of product and the quantity they require as well as their purchasing power. After a specific waiting time, customers become annoyed and reduce the player's reputation according to the customer's purchasing power. Additionally, customers vary in terms of payout, i.e., how much they add to the player's score once they are satisfied, which may decrease over time.

¹ The game UI and world visualization make use of several third-party assets; the background pattern is provided by Joel Klein (<http://subtlepatterns.com/>, CC-BY-SA 3.0). The tile images were created by Kenney Vleugels (<http://kenney.nl/>, CC0), and the UI elements are realized with the Space Game GUI Pack by Svetlana Shirokova (<http://graphicriver.net/item/space-game-gui-pack/9264290>).

In addition, the Game Design Document includes many features and proposals that could not be implemented as part of the prototype in this thesis due to time and resource constraints. These features include a story line set in Boston, strikes in overburdened factories, an online high score system, a branched level progression system, and the specification of several visually distinct areas to dictate the placement of buildings, e.g., based on the process groups TCPED (see section 1.1 on page 1).

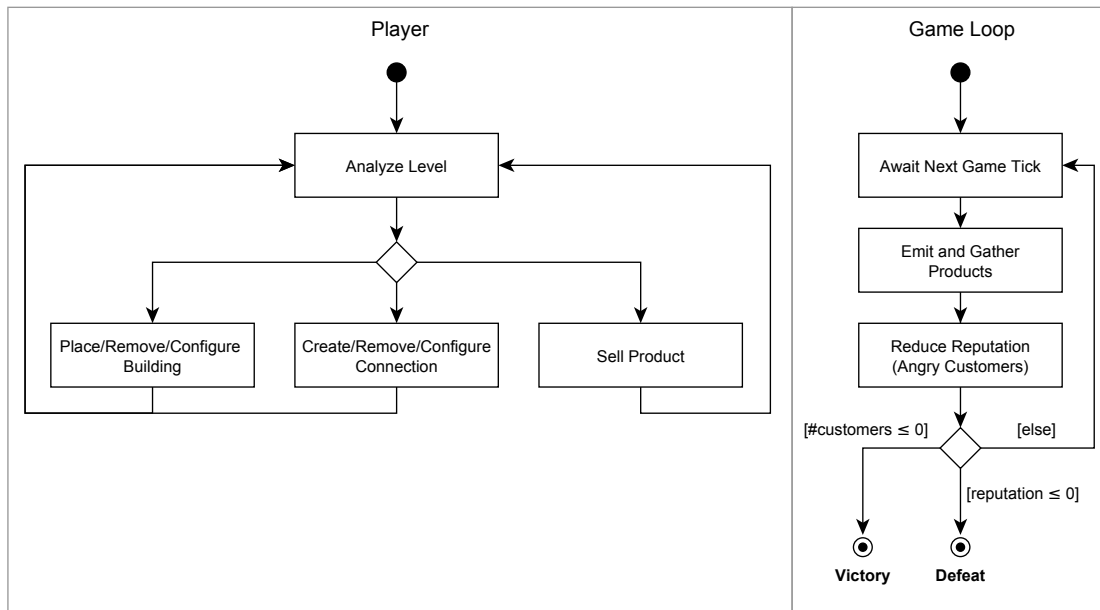


Figure 5.4: General gameplay (left) and game loop (right) of EXTRA.

Figure 5.4 illustrates the gameplay based on the aforementioned game elements and features. During a level, the main tasks of the player are to sell products collected at markets and to create, remove, and configure buildings and connections in order to establish production chains. At the same time, the game loop, which runs in the background, is responsible for emitting products from factories and gathering them at markets. Additionally, angry customers will reduce the player's reputation. The game loop also tests if the player has won or lost the level.

5.3 Architecture

As previously mentioned, the architectural design decisions resulted from the requirements specified in the Game Design Document and the results of the game engine analysis presented in chapter 4 on page 31.

Since online features, such as a high score systems and multiplayer support, are desired in the long term, the game includes a server-side component, which can also be used as the web server to deliver the game client itself over the internet or intranets.

To reduce switching costs for the game engine and UI library and to realize the modularization described in section 5.2 on page 40, the world state has been decoupled from its representation as suggested by the widespread MVC pattern. Additionally, the world state is realized as a single,

self-contained JavaScript object that can easily be serialized to JSON for persistence or to transmit it to the server or other clients.

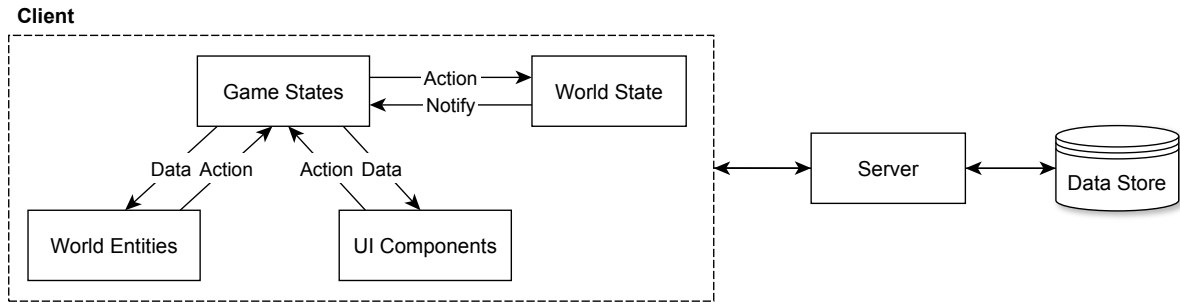


Figure 5.5: Architectural overview of EXTRA.

In accordance with the aforementioned architectural patterns, the chosen architecture is illustrated in fig. 5.5, and is described in detail in the remainder of this section.

5.3.1 Client-Server Model

As is usually the case for web-based applications, the game is divided into two components: a client and a server communicating over the HTTP protocol. This enables multiple, heterogeneous clients (e.g., mobile and desktop clients) to simultaneously play the game, and allows the server to centrally manage various data such as the scenario, high scores as well as recorded user actions. Furthermore, the server can potentially verify or be responsible for transitions of the world state to prevent cheating or to reduce the clients' processing requirements.

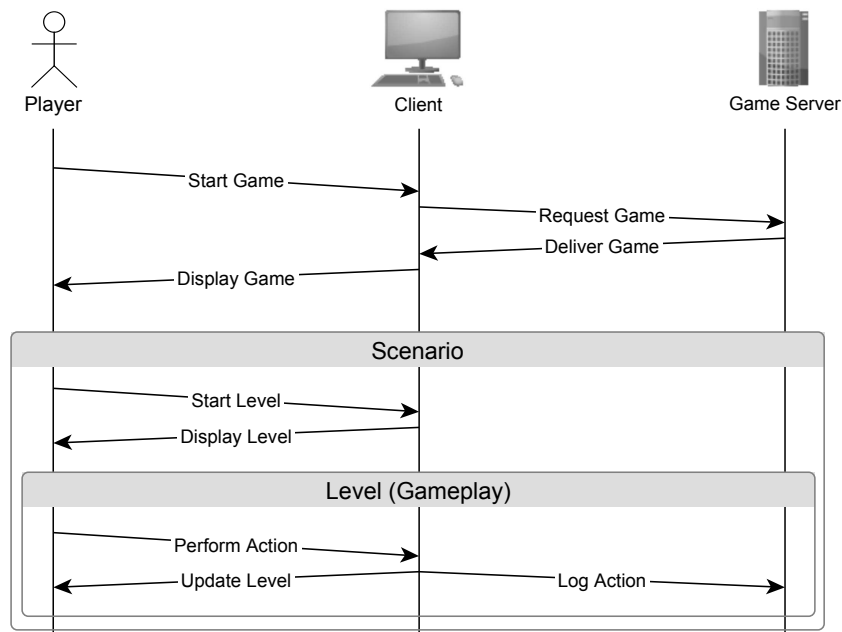


Figure 5.6: Client-Server communication in EXTRA.

As shown in fig. 5.6, the overall communication procedure can be described as follows: first, the client's browser requests the game client component via a specific URL (web server functionality).

The server then responds accordingly, at which point the client can request additional data, such as the scenario model and assets, before the game commences. During the course of the game, the client transmits various data including the user actions. Subsequently, the server can store or analyze these actions (logging functionality) for review. The client may also request additional data such as the list of high scores.

To facilitate development and debugging and to provide a stand-alone version that can be played without an internet connection, the game client can also run on its own without the server. In this form of deployment, however, the game is limited by the browser's security restrictions, e.g., it cannot arbitrarily read from or write to the file system without user interaction. Therefore, in this case, the scenario has to be embedded at build time.

5.3.2 Game World State

The world state representing the model is separated from the rest of the game. This entails that it can be used in isolation and does not depend on any other modules or components. The approach taken is inspired by (and realized with) the Redux JavaScript library (redux.js.org), which in turn is based on existing principles of functional programming, Command Query Responsibility Segregation (CQRS) [Fow11] and Event Sourcing [Fow05].

Essentially, the dynamics of the world state can be illustrated as:

$$state_n = reducer(state_{n-1}, action)$$

The serializable world state is expressed as `state` while `action` represents a serializable object describing the action (e.g., user action) similar to an event. The initial state of each level (`state_0`) is specified as part of the scenario. To progress the game, the `reducer` function applies transformations on the current state based on the given action to compute the next state.

Redux specifies the following fundamental principles [Abr16]:

1. *Single source of truth*: the state is stored as an object tree in a single, dedicated place.
2. *State is read-only*: the state is never mutated directly, but only by applying an action on the current state as illustrated above.
3. *Changes are made with pure functions*: reducers are implemented as pure functions, i.e., they are deterministic and free of side effects.

This approach yields several benefits, as both the state and the sequence of actions can be serialized to be stored locally or to be transmitted to the server at any time. By keeping track of some past `state_n` and the succeeding actions, all intermediate states can be recalculated at will. Additionally, since the reducer function is pure, it is relatively easy to unit test and it is simpler to comprehend and to debug because the model and its past mutations (resulting from actions) can be inspected, reverted, and re-applied (time-travel).

Consequently, it simplifies the implementation of the following common game features: undo and redo, logging and replay, server-side processing or verification of game logic, observing other players in real-time, and save games. Moreover, the ability to replay could especially be useful for after-mission rehearsals.

The serializability of the game state and the actions could also be useful to integrate EXTRA with other systems. To illustrate, the stream of user actions could be transformed to activities as specified

by the TinCanAPI (tincanapi.com) and subsequently be transmitted to a learning management system or learning record store. Thus, the user's learning progress could be recorded and analyzed in a standardized and interoperable way.

It should be noted, however, that *some* of these properties could also be achieved with a more conventional (e.g., class-based) approach that allows the (un-)serialization of the state at any given time, which may also be more familiar to many developers. Nonetheless, the functional approach was ultimately preferred, as the class-based alternative would not cover all the aforementioned properties without further efforts (e.g., serializable actions and pure functions for testability). In addition, JavaScript offers first-class support for functions, yet only added built-in class support as part of ES2015.

5.3.3 Game Client Architecture

The game client of EXTRA is realized according to the MVC paradigm, which entails that the model, controller, and view components are separated. In line with the results of the game engine analysis in chapter 4 on page 31, EXTRA incorporates both a game engine (Phaser) and a UI library (React). While the game engine handles the responsibilities of the controller, both the game engine and the UI library are responsible for displaying the view. The following section describes the individual components and their relationship to MVC (see fig. 5.5 on page 44).

World State The world state represents the model in MVC, and its realization is described in section 5.3.2 on page 45. A store is used to manage the current state by allowing other parts of the program to dispatch action objects and to subscribe to changes (observer pattern [Gam95]).

Game States Top-level game states are common features of game engines. Based on the state pattern [Gam95], they are often implemented as finite state machines, managing transitions between various states, e.g., loading state, options screen state, or play state. These top-level states are responsible for managing the world state store, setting up the view components, and passing data to the views by subscribing to the store. Moreover, they forward actions from the views to the store and manage the communication with the server. As such, they can be considered the controller in MVC.

World Entities Another common feature of game engines is the ability to establish a hierarchical scene graph using several entities (or display objects), which can be seen as a realization of the composite pattern [Gam95]. The entities are then used to visualize the game world. They receive data derived from the current game state and generate action objects based on user interactions. Hence, they represent part of the view in MVC.

UI Components Similar to world entities, the UI components are part of the view, receive data, and forward user actions. Moreover, they can also often be composed hierarchically. In contrast to the world entities, they visualize the game's UI which is layered on top of the game world.

The benefits of this architectural approach are a clear separation of concerns and the isolation of the model, which is easier to test and can potentially be used outside of the game client, e.g., on the server. Additionally, the data flows are clearly organized, making it easier to track state changes.

5.4 Design and Implementation

The development of the prototype involved several iterations (see fig. 5.7). After the game engine analysis presented in chapter 4 on page 31, a clickable prototype was implemented based on an early mockup from the Game Design Document using Phaser and React. Next, core gameplay elements and an isometric game world visualization were included. And finally, the UI was overhauled to improve usability and the visual appeal.

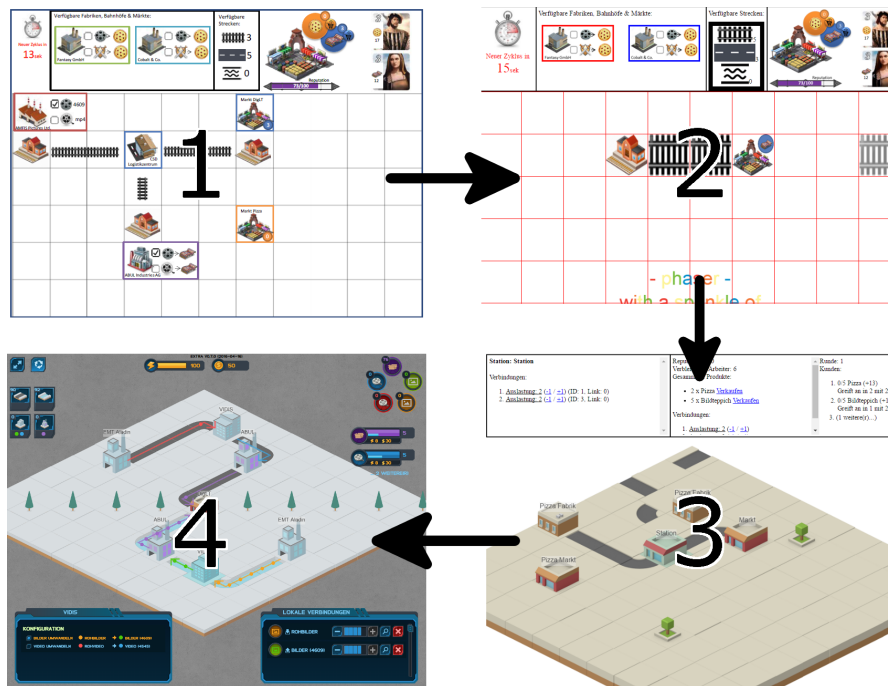


Figure 5.7: Stages of development: (1) early mockup from the Game Design Document, (2) first clickable prototype, (3) nearly feature-complete implementation, (4) final prototype.

The prototype is implemented as a single NodeJS module written in ES2015, containing both the client and server component. In addition, Webpack provides a development server and is responsible for the build process. These technologies are described in section 3.5 on page 25.

This approach has many benefits: it, for example, allows for the source code to be cleanly organized in different modules with explicit dependencies, and the client and the server components can easily share code, as they are both written in JavaScript. Additionally, Webpack allows for a flexible build process, (e.g., various assets can be embedded into the final bundle), making it possible to create stand-alone bundles of the game. Moreover, the implementation can easily utilize the modules published on the package repository of NodeJS (npm).

Since only a subset of the features specified in the Game Design Document was implemented in this prototype, this section focuses on the data structures of the scenario and the user actions (model in MVC), the game world visualization, the user interface (view in MVC), and the implementation of the server component. As for the game states (controller in MVC), the prototype only includes the *PlayState*, which is responsible for setting up the model and view and for realizing the communication between the two. The *PlayState* also manages the network communication between client and server.

5.4.1 Scenario Specification

One of the primary goals of EXTRA is to allow authors to easily specify scenarios and game levels in order to support a wide range of exercises. For reasons of interoperability, scenarios can be defined as a combination of game levels (JSON), translation files (JSON), images (various formats), and a machine-readable representation of a UML activity diagram (XML).

This approach allows different modeling languages to be transformed to the scenario format as long as a meaningful mapping can be found. For example, factory, process, and product names could be extracted from XML representations of UML activity diagrams to be integrated into the individual levels. In addition, Tiled Map Editor files (Tile Map XML, or TMX) files could be used for the initial placement of structures and their images.

While playing a scenario, the levels are progressed linearly, and one level is active at any given time. At the start of each level, the store holding the world state is initialized with the corresponding level data structure, i.e., the level specifications are used as the initial state $state_0$ (see section 5.3.2 on page 45).

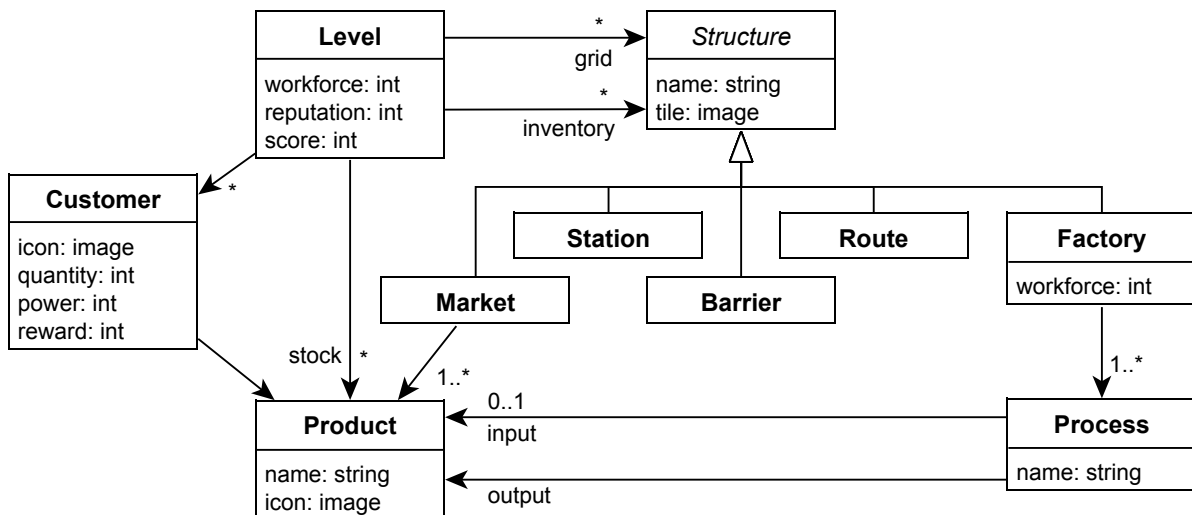


Figure 5.8: Model of the level data structure implemented as a JavaScript object and specified using a JSON file.

The static level structure is illustrated in fig. 5.8. At the top (i.e., the root of the level's JSON file), the remaining workforce and the player's current reputation and score are stored. Additionally, a series of customers (opponents) is specified, including their product requests, purchasing power, and the score awarded to the player once they are served. Furthermore, metadata, such as version numbers and author names, can be specified without affecting the gameplay.

The level structure also contains the 2-dimensional grid and the player's current inventory, both of which consist of a set of structures. Furthermore, each of these structures has at least a name and an image. Factories also specify a set of named process with their output and input products. Markets, on the other hand, are characterized by the products that they accept, while routes can also be of different route types (street, rail, or waterway), which was omitted for brevity.

In addition, as JSON does not support class inheritance, these relationships are realized using a `type: string` attribute. For internationalization (i18n), all displayed strings are specified as keys that

can be resolved using the translation files of the scenario. Images are resolved in a similar manner based on their file names.

Listing 5.1 shows an excerpt of a level specified in JSON this way.

```
1 {
2   "grid": [
3     [
4       null, null,
5       {
6         "type": "factory",
7         "name": "EMT Aladin",
8         "image": "aladin.png",
9         "workforce": 4,
10        "configurations": [
11          {
12            "output": "raw_pics",
13            "name": "Record Pictures"
14          }
15        ]
16      },
17      null, null, null, null
18    ]
19  ],
20  "allProducts": [
21    {
22      "id": "raw_pics",
23      "name": "Raw Photos",
24      "icon": "photo.svg"
25    }
26  ],
27  "workforce": 20,
28  "reputation": 100
29 }
```

Listing 5.1: Excerpt of a level specification in JSON.

5.4.2 Actions

As explained in section 5.3.3 on page 46, actions are used to progress the world state during the game. On a technical level, they are small and serializable JavaScript objects, containing a mandatory `type:string` attribute and additional attributes depending on their type. For better tooling support (e.g., code completion) and to avoid typographical errors, factory functions (action creators) are used to create action objects, instead of creating the objects on the fly.

The following section describes all actions implemented as part of the prototype using the signatures of their action creators:

- `tick()`: Advances the game by one time step. It is the only action that is automatically dispatched and not a direct result of user interaction. Its rate can be configured and defaults to 20 ticks per second. As a result of game ticks, customers may become annoyed and reduce the player's reputation. In addition, factories may emit products, and markets may collect them.

- `placeStructure(x, y, inventoryIndex)`: Places a structure specified by its index in the inventory on the grid at position (x, y) and decrements its quantity in the inventory.
- `removeStructure(x, y)`: Removes a structure on the grid at position (x, y) and increments its quantity in the inventory.
- `sellProduct(productId)`: Sells the product specified by its identifier (ID) to the next customer and removes it from the stock.
- `changeFactory(x, y, processIndex)`: Changes the factory's active process (i.e., input and output) at position (x, y) .
- `addFactoryWorkforce(x, y, n)`: Assigns n workers to the factory at position (x, y) , removing them from the pool of remaining workers.
- `removeFactoryWorkforce(x, y, n)`: Removes n workers from the factory at position (x, y) , adding them back to the pool of remaining workers.
- `connect(path, inventoryIndex)`: Establishes a connection between two buildings. Routes are automatically placed on the grid. The path is an array of (x, y) positions, and the inventory index specifies which route type to use.
- `removeConnection(connectionId)`: Removes a connection created with `connect()`.
- `incrementLoad(connectionId)`: Increments the current load of a connection, which determines the rate of products sent along the connection.
- `decrementLoad(connectionId)`: Decrements the current load of a connection.

The way game ticks are implemented essentially results in a “*fixed time step, variable rendering*” game loop approach [Nys14], where rendering and simulation are decoupled. To ensure smooth animations, the positions of moving entities are extrapolated based on their velocities and the time that has passed since the last game tick.

The reducer functionality to transform the current world state based on the actions listed above was developed using a test-driven approach, i.e., a set of unit tests was specified for each action type before their actual implementation in the reducer function. An example for such a test is provided in listing 5.2.

```

1 import level from './level.json';
2
3 describe('removeStructure', () => {
4   it('should remove the structure and add it to inventory', () => {
5     // arrange
6     const action = Action.removeStructure(0, 0);
7
8     // act
9     const state = reducer(level, action);
10
11    // assert
12    expect(state.grid[0][0]).toNotExist();
13    expect(state.structures[0].quantity)
14      .toBe(level.structures[0].quantity + 1);
15  });
16 });

```

Listing 5.2: A simple unit test for the functionality of the `removeStructure` action realized with the Mocha testing framework (mochajs.org).

5.4.3 World and User Interface

The view components comprise a mixture of Phaser display objects and containers (entities) and React components. The former are used to visualize the dynamic game world and are rendered to a 2D canvas, while the latter make up the user interface and are rendered as individual DOM elements.

Both of these component types are composed hierarchically and are implemented as ES2015 classes. While React components extend the `React.Component` class, Phaser entities inherit from `Phaser.Group` (containers used to establish parent-child relationships), `Phaser.Graphics` (to draw lines or shapes) or `Phaser.Plugin.Isometric.IsoSprite`. The latter class is provided by the Phaser Isometric plugin (rotates.org/phaser/iso/) to handle isometric projections.

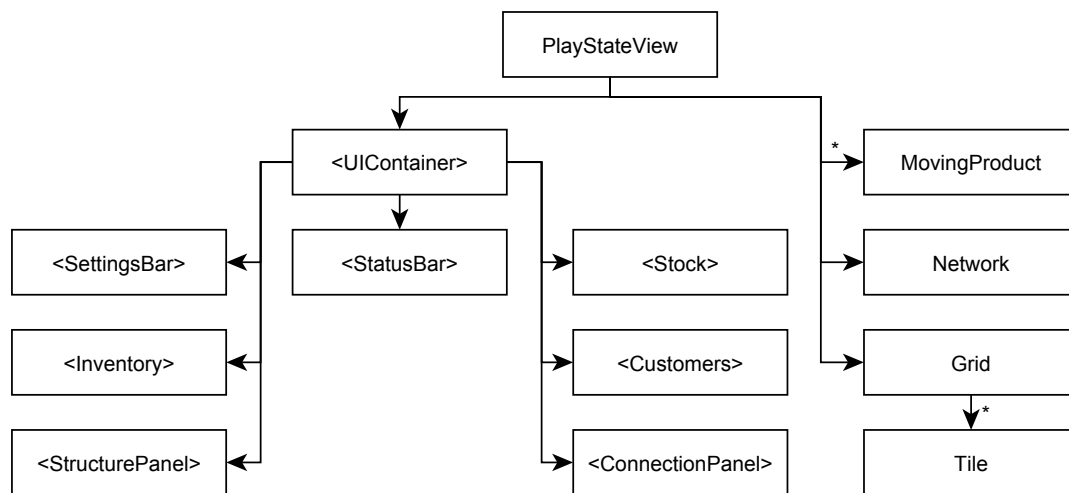


Figure 5.9: Model of the view hierarchy. React components are in angle brackets.

Figure 5.9 illustrates the component hierarchy and fig. 5.10 on the following page demonstrates the resulting visualization. The `PlayStateView` group is at the topmost level, and is set up by the `PlayState` itself. The `PlayStateView` is responsible for instantiating the `UIContainer`, the set of moving products (`MovingProduct`), the connection lines (`Network`), and the `Grid` which comprises various `Tile` entities, i.e., the structures present in the game world.

In addition, the connection lines are implemented using a series of line-drawing commands that are made available by `Phaser.Graphics`, while the moving products are realized using an object pool for performance reasons, which is a commonly used pattern in game programming [Nys14].

All UI components are descendants of the `UIContainer` and are displayed on top of the game world. The `SettingsBar` occupies the top-left corner, and currently features a button to activate full-screen mode and a toggle switch to enable or disable the visualization of connection paths. Beneath the settings bar resides the `Inventory`, used to select the route or building to place next.

The `StatusBar` is located at the top and displays the (remaining and total) workforce and the player's current reputation and score. The top-right corner features the current `Stock`, where players can see the amount of collected products and sell them to the customers. The `Customers` component, displayed below the stock, indicates the remaining times for the customers to become dissatisfied, their product requests, their purchasing powers, and their score rewards.

The `StructurePanel` and `ConnectionPanel` are at the bottom of the screen; these are both context-sensitive, which entails that their content varies depending on which structure is selected. If



Figure 5.10: Screenshot highlighting the main React UI components (framed in red).

none is selected, the structure panel disappears, and the connection panel exhibits *all* established connections, allowing the player to remove connections or to adjust their current load. If a structure or any other tile is selected, the connection panel only displays the connections that pass through this tile. The content of the structure panel depends on the type of the selected structure. For routes, it displays their current load, their maximum load (capacity), and their speed. When a market is selected, the panel exhibits the products which are accepted, and for factories, it allows users to switch between processes and adjust the workforce.

The game can be controlled with either a mouse or a touch screen, and was designed for a minimum resolution of 1024 by 700 pixels to support a wide range of desktop and tablet devices. As the UI elements are anchored relative to the screen corners, they will spread out on higher resolutions, leaving more space for the game world itself.

On devices with a low (logical) resolution, such as mobile phones, the game and UI will be scaled down which will likely render the game unplayable due to small fonts and input elements. To support these devices as well, the UI needs to be specifically optimized for them. As this is a common problem with web applications, there are many solutions available that could be implemented with React and existing web technologies, e.g., CSS media queries for adaptive layouts.

5.4.4 Server Implementation

As described in section 5.3.1 on page 44, the responsibilities of the server component are to serve the game client and the scenario, and to provide an endpoint for logging user actions. The server component is implemented using NodeJS with the express framework (expressjs.com) and can be used as a command-line application that merely requires the NodeJS binaries. These binaries can be downloaded as individual files for various platforms. Furthermore, several parameters, such as

the desired server port, can be configured using command-line arguments. Apart from external dependencies and the bundling of the scenario, the source code size of the server module is relatively small with about 150 lines.

Additionally, the game client itself is served at the root directory of the server. On a technical level, it consists of a HTML file (GET /, or GET /index.html) and a script file (GET /script.js); the latter is referenced by the HTML file and the browser will automatically download it. Assets, such as images, are embedded into the script file unless configured otherwise.

Futhermore, the scenario bundling occurs when the server is first started. The bundling works by processing a specific folder which is expected to contain a number of levels (JSON), translation files (JSON), and images (various formats), which are bundled into a single JSON file. This file is then exposed via HTTP GET /scenario to be downloaded and executed by game clients.

During the course of the game, the clients record all user actions and send them to the server on regular intervals via POST /log. For the sake of simplicity, the server does not use a full-fledged database solution, but simply stores the received actions in a series of JSON files. The client also passes metadata along with the actions, e.g., the user name, time stamps, and a randomly generated session identifier. The server also adds extra data such as the client's IP address.

Nonetheless, the server component is not required to play the game. To allow for faster development and to support the creation of a stand-alone client that can be run with just a web browser, it is possible to embed the scenario data into the game client and to prevent it from attempting to communicate with the server.

5.5 Application

Since NodeJS supports all prevalent desktop operating systems (e.g., Windows, MacOS, and Linux), EXTRA can be used in many situations and environments. Furthermore, EXTRA does not require any external dependencies or other resources from the internet after being bundled by Webpack. As a result, it can be distributed on any digital medium as a stand-alone .html file to be run locally on a computer (computer-based training) or it can be served via the internet (web-based training) or an intranet, similar to the SAR-Tutor learning system outlined in section 2.2 on page 12.

When using the server component, all user actions are recorded and the scenario can be changed without re-building the application itself. Although the primary use case of EXTRA is to train exercise participants in advance of the exercise in a web-based setting, these features could be useful for classroom environments and debriefings to further consolidate knowledge.

Currently, the authoring of scenarios involves the manual creation of JSON files. However, the process can be simplified using XML representations of UML activity diagrams (implemented as a proof of concept), which can be used to automatically insert factory processes and products accepted at markets. This information is derived from the specified activities and object flows. In the future, other machine-readable modeling formats, such as SysML and BPMN [Str16], could be supported and more game elements could be covered this way. Furthermore, the integration of Tile Map XML (TMX) files could facilitate the authoring of game levels. In addition to adapting the scenario, certain constants used in the game (e.g., the game's tick rate and the speed and capacity of the different route types) are stored in YAML configuration files and can thus be easily changed.

As previously explained, the game can be played on a wide range of devices including desktop computers, laptops, and tablets, as long as they have a modern browser and a certain physical screen size. Additionally, as internationalization (i18n) techniques were employed during the development, the

user interface and scenario elements can be localized using JSON or YAML files to make the game accessible to many users. Currently, the game includes localized strings in German and English, yet more languages can be included. The same mechanism can also be used to support different terminologies, e.g., with different degrees of abstraction regarding technical terms.

To ease the handling of the server component, which is realized as a command-line application, it would be possible to include a graphical user interface (GUI) using technologies such as NW.js (nwjs.io) or electron (electron.atom.io). These technologies are able to package NodeJS modules along with the Chromium web browser into a single executable file. Moreover, the same solutions could be used for the game client to remove the need for a web browser. Similar technologies exist for mobile devices (e.g., Cordova, cordova.apache.org) that make it possible to package web applications so that users can install and consume them like native ones.

6

Evaluation

The overall objectives of the pilot user study are to assess the technical functionality of the EXTRA prototype presented in chapter 5 on page 39, to evaluate its general usability and acceptance, as well as its effectiveness in conveying its learning contents. The study was not conducted with the target audience of the game (i.e., military personnel and other exercise participants), as this was beyond the scope of this thesis.

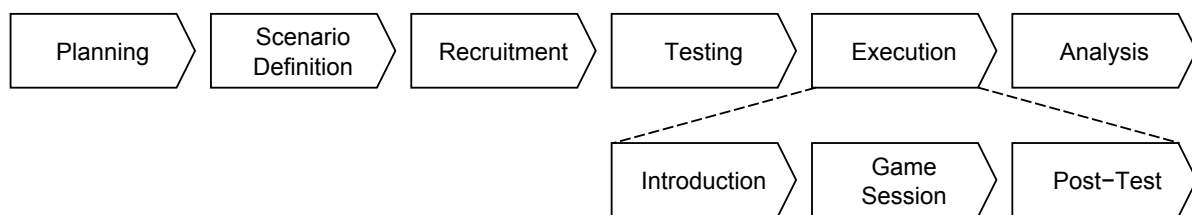


Figure 6.1: Outline of the evaluation approach.

The approach taken is illustrated in fig. 6.1. After planning the study, an imaginary exercise scenario was created based on real-world systems. Afterwards, the participants were recruited. Then, the feasibility of the setup was tested before the actual execution of the study. Finally, the results could be analyzed. Planning started in February 2016, and the execution took place in April 2016.

The study was initially intended to be conducted online without direct tutor support. However, early tests showed the need for assistance given the short designated play time of about 10 minutes, which was in many cases insufficient to properly accustom the participants to the game UI and mechanics. As a result, the study was conducted offline, so that the participants could be guided through the evaluation procedure and so that they could ask questions during the game sessions. Moreover, the levels specified as part of the scenario were simplified and the game mechanic of assigning workers to factories was removed entirely.

Based on the aforementioned objectives and the envisaged level design, the following hypotheses were formed and investigated using self-reporting measures and recorded user actions:

- **H1.** “Participants are able to complete the scenario without any severe technical failures.”
- **H2.** “Participants need more time to complete levels as they progress through the scenario.”
- **H3.** “Participants find the game concept to match the previously presented scenario.”

- **H4.** “Participants find the game easy to control.”
- **H5.** “Participants are able to recall the goal, structure and interlinking of the (sub-)systems of the scenario.”

This chapter is structured as follows: section 6.1 introduces the employed scenario. The design and procedure of the evaluation are described in section 6.2 on page 58. The demographics of the participants are summarized in section 6.3 on page 60, and the results of the evaluation are presented in section 6.4 on page 60 and discussed in section 6.5 on page 62.

6.1 Scenario

For the evaluation, an imaginary NATO exercise scenario was created and integrated into the game. It was called “Spectral Tiger” and was introduced to the participants before the game session.

Based on the TCPED model (see section 1.1 on page 1) illustrated in fig. 6.2, the overall objective of the scenario is to record, process and exploit images in order to produce actionable intelligence. The selected systems, data flows and processes can be seen in fig. 6.3 on the facing page. A clear distinction was made between pictures and videos to allow for different production chains and flows within the game. The model was also shown to the participants as part of the introduction.

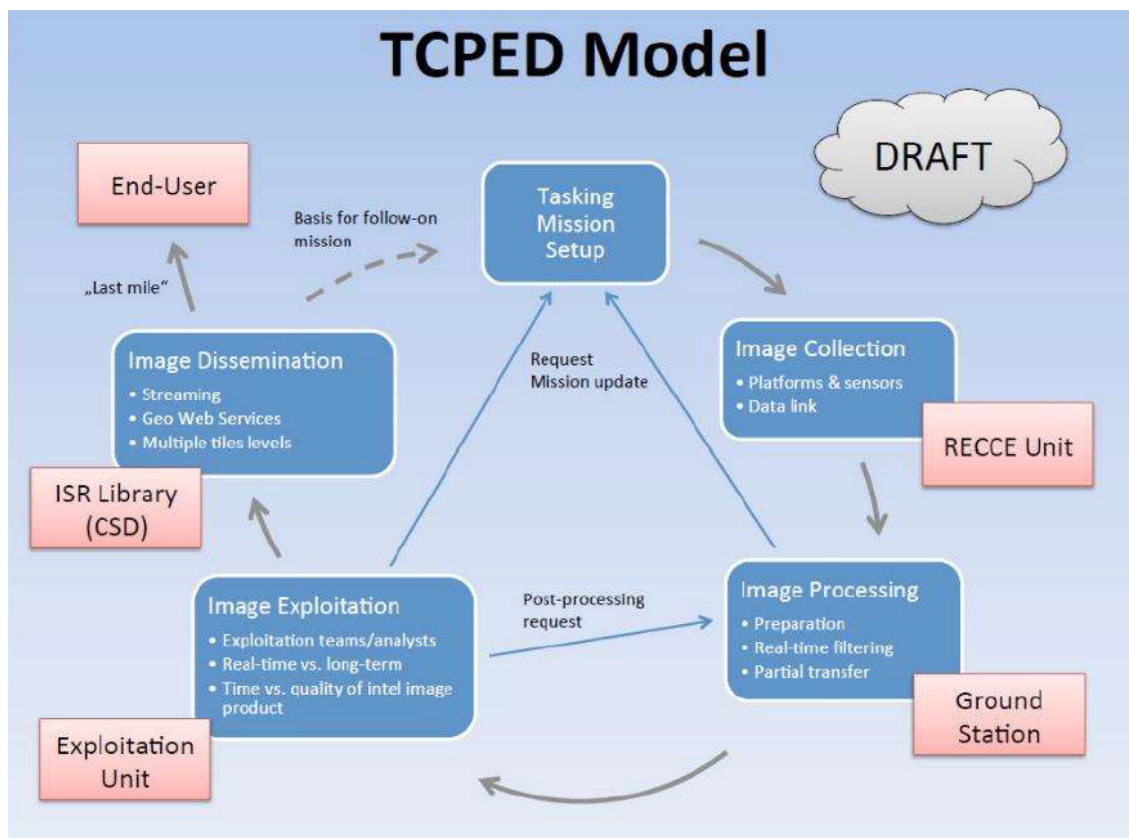


Figure 6.2: TCPED model which formed the basis of the scenario (Source: [Gem13]).

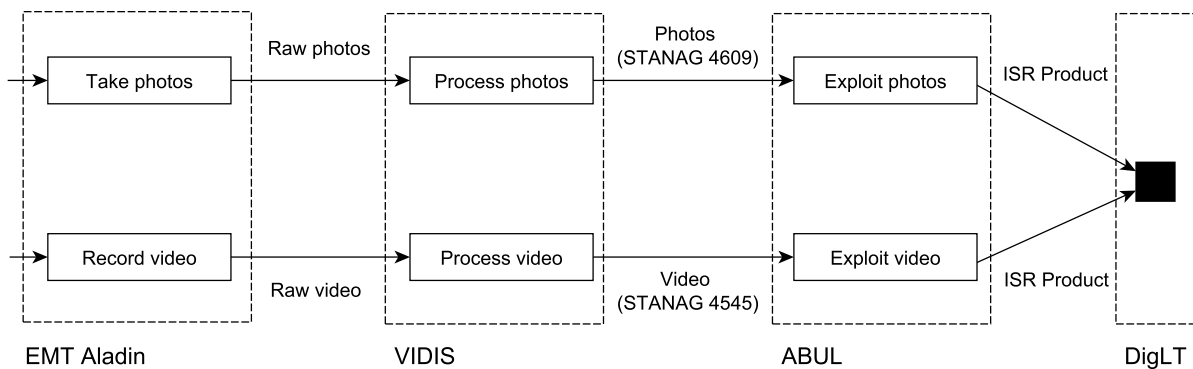


Figure 6.3: The JISR process used for the scenario and embedded into the game.

This scenario name “Spectral Tiger” was chosen because the word “spectral” relates to light waves and colors, which are captured by images, and because the word “tiger” is often used in military contexts; for instance, the NATO Tiger Association comprises air forces from multiple NATO member states. Additionally, “Spectral Tiger” is a reference to a virtual mount in the game World of Warcraft.

The following section describes the individual systems used in the scenario:

EMT Aladin stands for *abbildende luftgestützte Aufklärungsdrohne im Nächstbereich*, and is a miniature unmanned aerial vehicle (UAV) surveillance system developed by the German company EMT (emt-penzberg.de). Aladin is used by multiple NATO member states [EMT09].

VIDIS is a *video distribution system* made by M4Com System GmbH (m4com.de). It is able to process, transcode, and distribute various data streams to consumers. It can also correct errors to a specific degree, and is able to generate STANAG-compliant output streams [Kno11].

ABUL provides a series of computer vision algorithms for image exploitation to human interpreters and was developed by the Fraunhofer Institute of Optronics, System Technologies and Image Exploitation (Fraunhofer IOSB, iosb.fraunhofer.de). It was successfully tested by the German Army (Bundeswehr) [Hei10] and is compliant with STANAG specifications.

DigLT is a digital situation table (*Digitaler Lage Tisch* in German) that allows a team of experts and decision-makers to cooperatively analyze situations by providing a multi-user and multi-display environment [Wag11]. Similar to ABUL, DigLT was developed by the Fraunhofer IOSB and complies with STANAG specifications.

The actual task of the game was to place, connect, and configure these systems to serve the customers’ requests in a total of three levels. Therefore, the systems EMT Aladin, VIDIS and ABUL were integrated as factories with appropriate processes, and DigLT was represented by a market building. To reduce the complexity of the first two levels, the overall process was shortened, i.e., the systems VIDIS and ABUL were only presented in later levels, and the DigLT system also accepted products other than the final intelligence products.

The design of these three levels with their designated solutions is shown in fig. 6.4 on the next page and described in the following:

1. The first level allowed players to familiarize themselves with the game and its interface. The players simply had to connect the two pre-placed buildings (EMT Aladin and DigLT) to gather raw videos and sell them to the one customer present in this level.

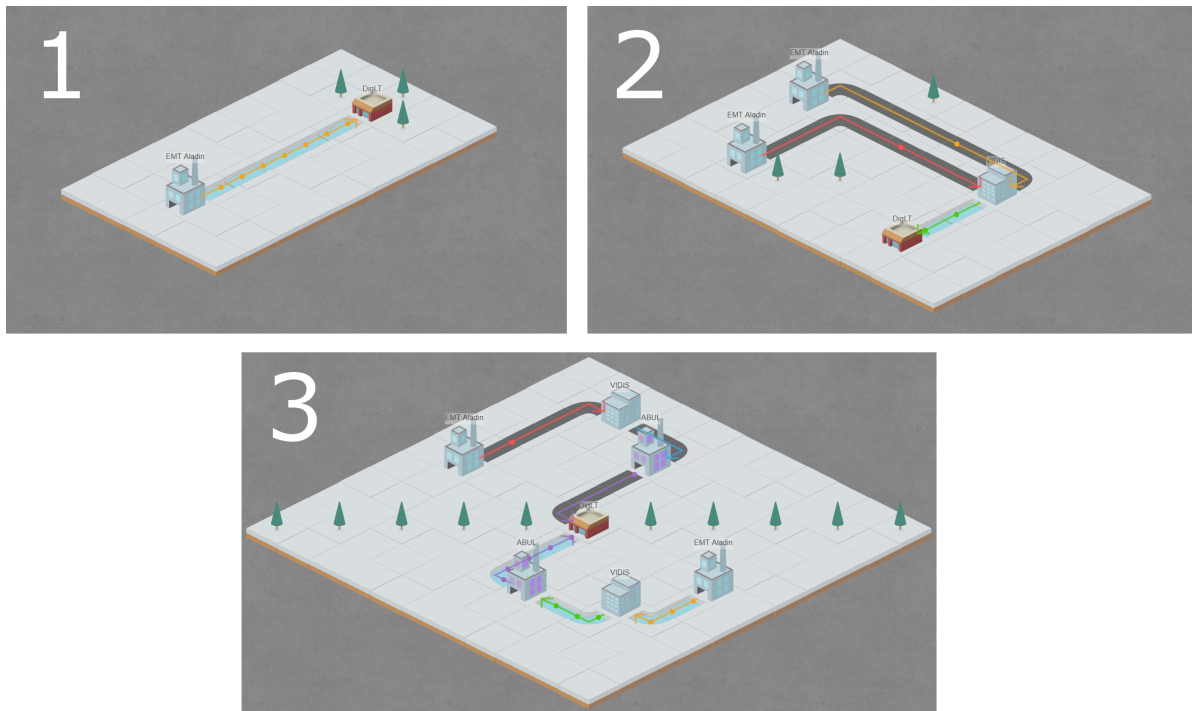


Figure 6.4: The three game levels of the scenario.

2. In the second level, the players had to place the VIDIS system on the grid, connect it to the two EMT Aladins (one for videos and one for photos), and forward the products to the DigLT system. Since the customers demanded different products (STANAG-compliant photos and videos), the players had to reactively switch between the two processes available in the VIDIS system. Alternatively, they could build two VIDIS systems, which was complicated, however, by the layout of the level, the pre-placed barriers (trees) and the limited amount of routes.
3. The last level involved the gradual setup of two complete processes comprising EMT Aladin, VIDIS, ABUL and DigLT. Again, two EMT Aladins were pre-placed on different sides of the levels, which were separated by a row of trees with the DigLT target system in the center. Due to varying customer requests, the players first had to produce and gather STANAG-compliant photos and videos, and then extend the production chain by incorporating the ABUL system to produce intelligence products. To give players a hint about the solution, VIDIS was pre-placed on one side, and ABUL on the other. Additionally, the players could (effectively) use an unlimited amount of routes.

6.2 Design and Procedure

The evaluation is designed as an uncontrolled user study without a pre-test. As illustrated in fig. 6.1 on page 55, the general procedure can be described as follows:

1. **Introduction:** At the start of each evaluation session, the participants were briefly introduced to the objective and general concept of the game and the employed scenario. This information

was printed on paper and given to the participants. They were allowed to ask questions and take as much as time as they needed. Afterwards, they were introduced to the gameplay and user interface by watching a 90 seconds-long video. Again, they were allowed to ask questions, and to pause or replay at any time.

2. **Game Session:** After the introduction phase, the participants started playing the scenario comprising three levels. The levels were designed for roughly 10 minutes of total play time. Various Windows-based desktop computers and laptops as well as iOS tablets were used for this purpose, while either Chrome or Firefox was employed as the web browser. During the game, the participants were allowed to ask questions. The game was served from a remote server to record all user actions. Figure 6.5 shows one of the participants playing on a tablet.
3. **Post-Test Questionnaire:** Immediately after finishing the scenario, the participants had to fill out a questionnaire, containing questions about the game concept, the game's usability, the scenario (test of knowledge), and their demographic data.

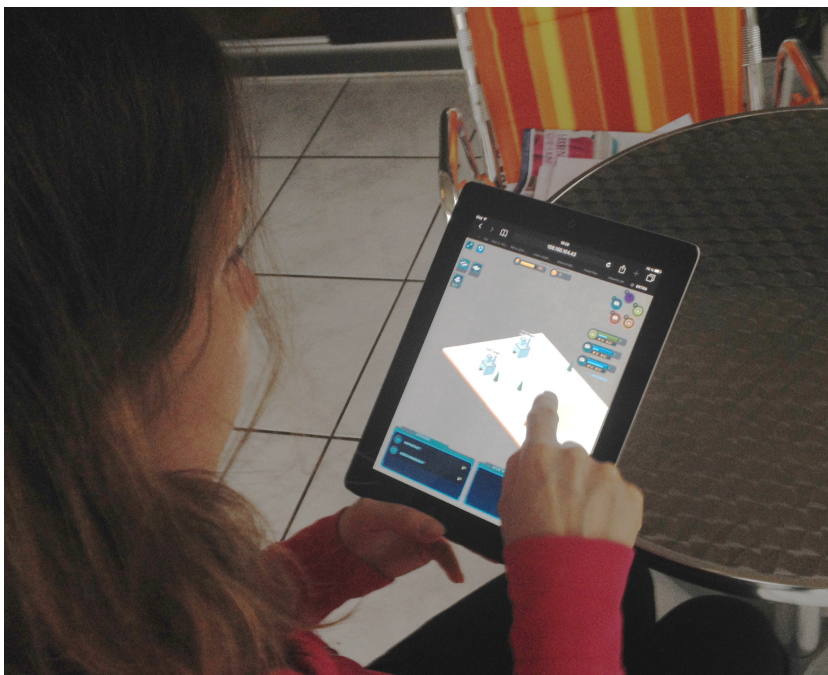


Figure 6.5: One of the participants playing EXTRA on an Apple iPad.

The questionnaire was created based on the formulated hypotheses and the scenario described in section 6.1 on page 56. The translated questions can be found in table 6.1 on the next page, while both the original introduction document and the questionnaire (in German) can be found in appendix B on page 89.

The questions regarding the game concept and usability used a five-point Likert scale [Lik32] ranging from 1: *Strongly disagree* to 5: *Strongly agree*, while the knowledge questions could be answered with *Yes*, *No* or *Uncertain*. The usability questions were derived from the *System Usability Scale* [Bro86].

In addition, the questionnaire included free-text fields for further comments about the concept and usability as well as demographic questions about the participants' age and gender.

Table 6.1: Contents of the questionnaire.

Game Concept	Q1	I like the concept of the game.	
	Q2	The concept matches the scenario introduced earlier.	
	Q3	The game's difficulty was appropriate.	
	Q4	The duration of the game was appropriate.	
Usability	Q5	I think that I would like to play this game frequently.	
	Q6	I thought the game was easy to control.	
	Q7	I would imagine that most people would learn to play this game very quickly.	
	Q8	I needed to learn a lot of things before I could get going with this game.	
Knowledge	Q9	The system CSD was part of the scenario.	<i>no</i>
	Q10	The system DigLT was part of the scenario.	<i>yes</i>
	Q11	The UAV system EMT Luna was part of the scenario.	<i>no</i>
	Q12	The system VIDIS was used to convert raw pictures and videos to (STANAG) standard formats.	<i>yes</i>
	Q13	The overall goal of the scenario was the exploitation of images.	<i>yes</i>
	Q14	The system ABUL could directly process the pictures and videos produced by EMT Aladin to create intelligence products.	<i>no</i>
	Q15	The system ABUL was part of the scenario.	<i>yes</i>
	Q16	The system VIDIS was mainly used to take photos and record videos.	<i>no</i>

6.3 Participants

After defining the scenario, the participants could be recruited to conduct the evaluation. Since the inclusion of the target audience was not intended, the participants were mostly friends and acquaintances (convenience sampling [Mar96]).

In total, $n = 9$ participants were recruited. Five of them were female and four male. Their ages ranged from 27 to 61 years with an average of 40 years. During the evaluations, it could be observed that all the participants had experience in using computers and had played computer games before.

6.4 Results

In support of **H1** (*participants are able to complete the scenario without any severe technical failures*), no technical problems occurred during the evaluation, and all user actions were successfully recorded on the server.

The individual level completion times derived from the recorded data can be found in fig. 6.6 on the next page. With 1 exception (**P6**), all the participants finished the scenario by completing all levels. On average, level 1 took *1 minute and 39 seconds*, level 2 took *9 minutes and 45 seconds*, and level 3 took *9 minutes and 36 seconds*. Additionally, 4 out of the 8 participants who completed both levels required more time for level 2 than level 3. Therefore, **H2** (*Participants need more time to complete levels as they progress through the scenario*) is not supported.

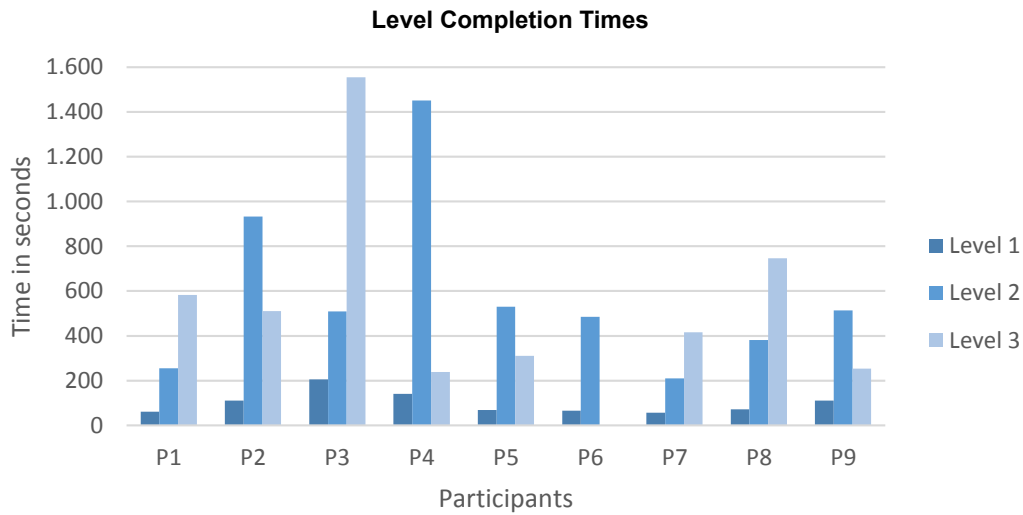


Figure 6.6: Recorded level completion times.

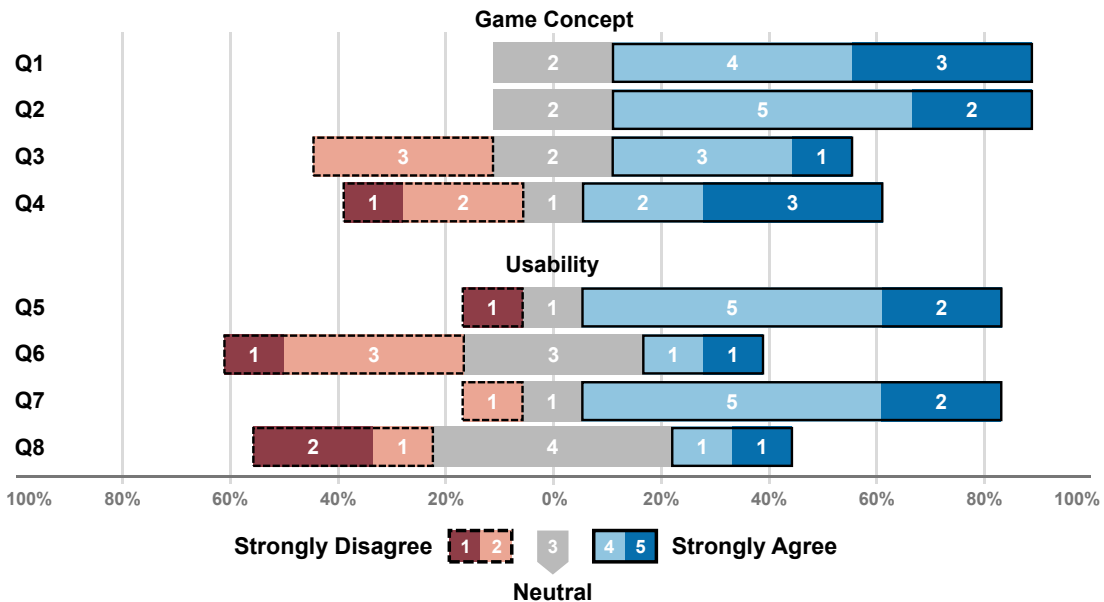


Figure 6.7: Results of the questions related to the game concept (top) and usability (bottom).

As can be seen in fig. 6.7, all the participants either found the game concept to match the given scenario or took a neutral stance (Q2). Thus, **H3** (*Participants find the game concept to match the previously presented scenario*) is supported by these results. Similarly, most participants liked the game concept in general (Q1), but opinions were mixed regarding the game’s difficulty (Q3) and duration (Q4).

The results of the questions related to the usability of the game and its controls are visualized in fig. 6.7. **H4** (*Participants find the game easy to control*) is not supported by these results, as only 2 participants responded positively, while 4 responded negatively and 3 participants abstained (Q6). The remainder of the questions related to usability (Q5, Q7, and Q8) yielded more positive results considering that Q8 is a negative item.

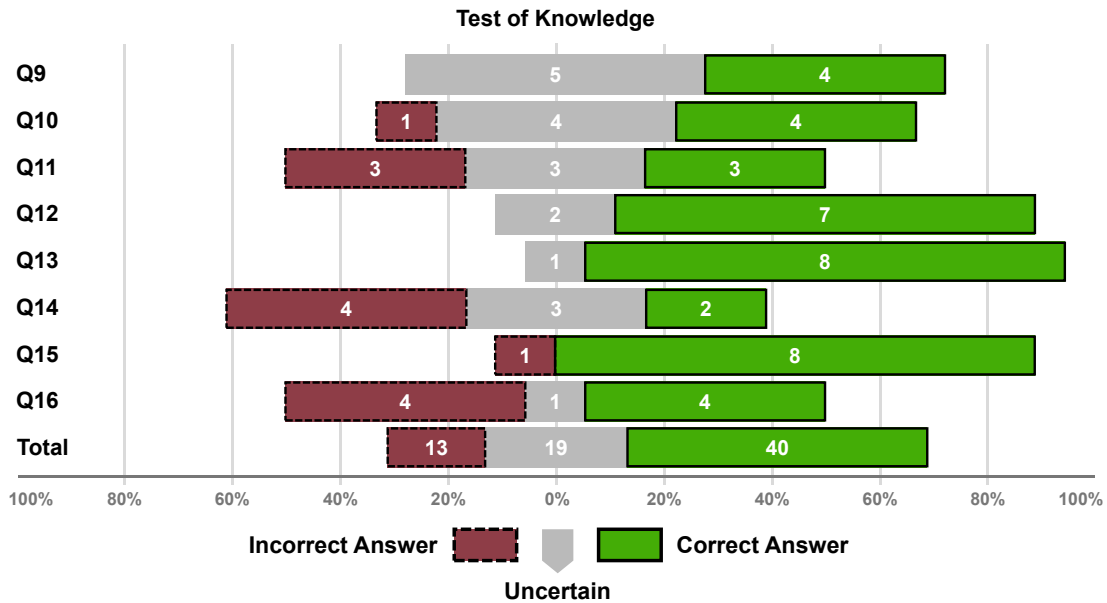


Figure 6.8: Results of the test of knowledge.

As illustrated in fig. 6.8, the majority of multiple choice knowledge questions were answered correctly. Thus, **H5** (*Participants are able to recall the goal, structure and interlinking of the (sub-)systems of the scenario*) is supported by this study. While 7 participants answered more questions correctly than incorrectly, 1 participant was able to answer all the questions correctly, and 1 participant answered an equal amount of questions correctly and incorrectly.

On the free-text fields of the questionnaire, 2 participants mentioned that the in-game objective of serving customers was not obvious to them. Also, some participants found the customer system and the way buildings had to be selected before placing them on the grid to be difficult to comprehend. Further, 3 participants suggested the need for a more comprehensive gameplay introduction, and 1 participant proposed to allow players to slow down the game. Similarly, 1 participant suggested to not impose a time limit (through customers) on the first level.

6.5 Discussion

Even though the target audience was not included, the evaluation featured a diverse set of participants as is evidenced by the fact that one participant completed the scenario in roughly 11 minutes, while another required more than 37 minutes, much longer than the initially planned 10 minutes. Also, the former participant was able to establish the correct production chains even before the customers requested the corresponding products, while the latter struggled to understand the controls. The same participant also commented that he rarely uses computers in day-to-day life.

During the game session, several participants said that they felt stressed and were frustrated by the game after losing a level, including the first one, which could be completed in roughly 30 seconds and with few interactions.

These observations emphasize the need for an adaptable (or adaptive) difficulty system due to the diversity of the target audience. In the simplest case, this could be realized in the form of multiple,

user-selectable difficulty levels to reduce the reputation penalty caused by dissatisfied customers or to increase the total reputation, which would effectively increase the time limit.

The study also revealed several areas for improvement regarding the controls of the game. For example, many participants initially struggled to establish connections between two structures which involved dragging a line between two buildings. Currently, the player has to start and end dragging exactly at the tiles of the two buildings with no intermediate feedback about the validity of the path. This interaction was further complicated by the diamond shape of the tiles and by the fact that high buildings visually extend beyond the tiles borders, causing players to unintentionally start dragging one tile too high.

Another interaction mechanic that confused many participants was the way buildings and routes had to be selected in the inventory UI in order to place buildings or to create connections. It seemed that many participants expected the selection to be automatically removed after they had made use of it. In an attempt to re-select the inventory item, they deselected it instead. Some players also attempted to use drag and drop for placing buildings, although, in that case, they seemed to have learned and adjusted quickly. One possible way to mitigate this issue would be to make the selected item more prominent, e.g., by highlighting it in a very different color instead of just making it brighter.

Overall, the test of knowledge yielded favorable results, supporting the feasibility of the educational approach taken by EXTRA. The three questions that caused the majority of incorrect answers were *Q11*, *Q14* and *Q16*. While *Q11* might have confused the participants by mentioning EMT Luna (instead of EMT Aladin), *Q14* might have been difficult because the ABUL system was only present in the last level. However, since both *Q14* and *Q16* relate to processes and data flows, further efforts might be required to make these elements clearer and more comprehensible. Based on the small scope and informal nature of this study, however, such conclusions should be treated with caution.

Several participants struggled to understand game mechanics, such as the customer system, and the overall objective of the game. An engaging story, sophisticated feedback and guidance could help mitigate these issues. For example, sounds could make it more obvious when customers deduct reputation, and some sort of notification system could inform players about invalid connections or factory configurations.

After completing the scenario, however, a number of participants said that they had finally understood the game and its controls, and that they would perform much better if they played again, suggesting that the play time was too short for the participants to become familiar with the game.

Conclusion

This thesis presents the design, prototypal implementation, and evaluation of a modular and web-based serious game called Exercise Trainer (EXTRA) based on pre-specified concepts. The primary goal of EXTRA is to help prepare participants of large-scale, multi-national training scenarios by familiarizing them with the employed systems and their interactions.

As military systems and processes are becoming increasingly complex and thus more difficult to comprehend, large-scale exercises for NATO Joint Intelligence, Surveillance, and Reconnaissance (JISR) are regularly performed; however, these exercises are often obstructed by insufficient staff preparedness. For this reason, emphasis needs to be placed on the training and education of military personnel and other participants to ensure the effectiveness of such exercises. Yet, no preparation tools exist for this purpose.

The presented solution exploits the concepts of digital game-based learning to intrinsically motivate the participants and to deliver an engaging and playful learning experience. To this end, EXTRA is designed as an economic simulation game that allows for real-world systems to be embedded into the game on an abstract level. Furthermore, by using the web as an application platform, EXTRA can be used on a variety of devices with just a web browser and without a complicated installation process. Additionally, due to the client-server architecture, user actions can be recorded in real-time and game contents can be centrally managed. Hence, the employed scenario can easily be adapted to changing requirements.

For the realization of EXTRA, the thesis started with a state-of-the-art analysis to identify the previous work in related fields and the concepts that have been employed. Afterwards, a subjective value benefit analysis was performed to systematically select the HTML5 game engine most suited to fulfill the requirements of the game. This analysis also incorporates two user interface (UI) libraries to ease the implementation of the static UI controls present in the game by utilizing existing web technologies such as HTML and CSS. According to the research findings, the combination of the Phaser game engine and the React UI library was deemed most appropriate and was therefore chosen for the implementation of EXTRA.

After selecting the game engine and UI library, EXTRA's architecture and design were specified. The game client's architecture is based on the MVC pattern and makes use of concepts from functional programming to managing user actions and the game's world state, both of which are represented by JSON data structures. Thus, this approach makes it easier to test the game logic in isolation and to realize common game features such as undo and redo, logging and replaying of game sessions, and

to save and restore the player's progress. Moreover, the ability to replay and analyze game sessions could especially be useful for after-mission rehearsals.

In addition, the implementation of EXTRA was succeeded by a pilot study to test the technical functionality, the general usability and acceptance, as well as the game's effectiveness in conveying its learning contents. Therefore, an imaginary exercise scenario based on real-world systems was specified and embedded into the game in the form of three game levels. While the course of the study revealed some areas for improvement in terms of usability, which may partially be attributable to the short designated play time, the results support the game's acceptance and its effectiveness in terms of learning. Furthermore, no technical problems were encountered during the study.

Although the study did not include any military personnel or other potential participants of military JISR exercises, it featured a diversity in terms of age, gaming affinity, and gaming experience, which is also expected to apply to the target audience. Due to this diversity, the results of the study emphasize the need for a difficulty system to keep each player in his or her individual flow zone. In the simplest case, this could be realized in the form of multiple, user-selectable difficulty levels to increase the implicit time limit; more sophisticated solutions could involve the dynamic adjustment of the difficulty (Dynamic Difficulty Adjustment) [Hun04].

Additional future work could be dedicated to the game features specified in the Game Design Document that were not realized as part of the prototype due to time and resource constraints. These features include a story line set in Boston, strikes in overburdened factories, a branched level progression system, and the specification of several visually distinctive areas that resemble process groups. Based on the insights gained from the pilot study, especially the story line should prove valuable to further engage the players, to gradually introduce them to the game mechanics, and to make the overall goals of the game clearer.

Furthermore, social interactions, such as competition and collaboration, could be promoted in the future by employing common game mechanics, including online multiplayer and high score systems, to further increase motivation. These mechanics can, for example, be realized using web technologies.

Finally, as the scenarios and user actions are represented by JSON data structures, the prototype offers great potential in terms of integration. While the support for UML activity diagrams is already implemented as a proof of concept, other modeling formats, such as SysML and BPMN, could be integrated as well [Str16]. In addition, Tile Map XML (TMX) files could be employed to facilitate the authoring of the individual levels. The user actions, on the other hand, could be transformed to activities as specified by the TinCanAPI (tincanapi.com) and subsequently be transmitted to a learning management system or learning record store. This would allow for the learner's progress to be recorded and analyzed in a standardized and interoperable way.

To conclude, and to return to the research questions raised in section 1.3 on page 4, the results of this thesis support the feasibility and effectiveness of EXTRA and its concepts as a tool to train exercise participants. Additionally, EXTRA holds a promising future with regards to feature enhancements, interoperability, and versatility in its application.

Bibliography

- [Abr16] ABRAMOV, Dan: Three Principles | Redux (2016), URL <http://redux.js.org/docs/introduction/ThreePrinciples.html>, last visited April 27, 2016
- [Alv08] ALVAREZ, Julian and MICHAUD, Laurent: *Serious Games – Advergaming, edugaming, training and more* (2008), URL <http://www.ludoscience.com/EN/diffusion/285-Serious-Games.html>
- [Bar15] BARON, Pete: Phaser 3 Development: Development Log 9 (2015), URL <http://phaser.io/labs/phaser3-development-log-09>, last visited April 27, 2016
- [Bow13] BOWERS, Clint A.; SERGE, Stephen; BLAIR, Lucas; CANNON-BOWERS, Janis; JOYCE, Rachel and BOSHACK, James: The Effectiveness of Narrative Pre-Experiences for Creating Context in Military Training. *Simulation & Gaming* (2013), vol. 44(4): pp. 514–522
- [Bre10] BREUER, Johnannes and BENTE, Gary: Why so serious? On the Relation of Serious Games and Learning. *Eludamos. Journal for Computer Game Culture* (2010), vol. 4(1): pp. 7–24, URL <http://www.eludamos.org/index.php/eludamos/article/view/vol4no1-2>
- [Bro86] BROOKE, John: SUS – A Quick and Dirty Usability Scale. *Usability evaluation in industry* (1986), vol. 189(194): pp. 4–7
- [Che07] CHEN, Jenova: Flow in Games (and Everything Else). *Communications of the ACM* (2007), vol. 50(4): pp. 31–34
- [Chi88] CHIN, John P; DIEHL, Virginia A. and NORMAL, Kent L.: Development of a Tool Measuring User Satisfaction of the Human-Computer Interface (1988)
- [Chi15] CHIN, Yee Cheng: How to Implement AAA Game UI in HTML and JavaScript. *Game Developers Conference 2015* (2015)
- [Cre10] CREUTZFELDT, Johan; HEDMAN, Leif; MEDIN, Christopher; HEINRICHS, Wm. LeRoy and FELLÄNDER-TSAI, Li: Exploring Virtual Worlds for Scenario-Based Repeated Team Training of Cardiopulmonary Resuscitation in Medical Students. *Journal of Medical Internet Research* (2010), vol. 12(3), URL <http://www.jmir.org/2010/3/e38/>
- [Cre12] CREUTZFELDT, Johan; HEDMAN, Leif and FELLÄNDER-TSAI, Li: Effects of Pre-Training Using Serious Game Technology on CPR Performance – An Exploratory Quasi-Experimental Transfer Study. *Scandinavian Journal of Trauma, Resuscitation and Emergency Medicine* (2012), vol. 20: p. 79, URL <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3546885/>

- [Cro08] CROCKFORD, Douglas: *JavaScript: The Good Parts*, O'Reilly Media, Inc. (2008)
- [Csi91] CSIKSZENTMIHALYI, Mihaly: *Flow: The Psychology of Optimal Experience*, Harper Perennial (1991)
- [Dah09] DAHL, Ryan: Node JS (2009), URL <https://www.youtube.com/watch?v=EeYvF17li9E>, last visited April 27, 2016
- [Dav89] DAVIS, Fred D.: Perceived Usefulness, Perceived Ease of Use, and User Acceptance of Information Technology. *MIS Quarterly* (1989), vol. 13(3): pp. 319–340
- [DeB16] DEBILL, Erik: Modulecounts (2016), URL <http://www.modulecounts.com/>, last visited April 27, 2016
- [Det11] DETERDING, Sebastian; DIXON, Dan; KHALED, Rilla and NACKE, Lennart: From Game Design Elements to Gamefulness: Defining "Gamification", in: *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, MindTrek '11, ACM, New York, NY, USA, pp. 9–15
- [Dja11] DJAOUTI, Damien; ALVAREZ, Julian; JESSEL, Jean-Pierre and RAMPNOUX, Olivier: Origins of Serious Games. *Serious Games and Edutainment Applications* (2011): pp. 25–43
- [ECM15] ECMA INTERNATIONAL: Standard ECMA-262 - ECMAScript Language Specification (2015), URL <http://www.ecma-international.org/publications/files/ECMA-ST/ECma-262.pdf>, last visited April 27, 2016
- [Elk87] ELKERTON, Jay: Online Aiding for Human-Computer Interfaces. *Handbook of Human-Computer Interaction. M. Helander* (1987)
- [EMT09] EMT PENZBERG: ALADIN Mini-Luftaufklärungssystem (2009), URL http://www.emt-penzberg.de/uploads/media/ALADIN_de_01.pdf
- [Fau14] FAUTUA, David; SCHATZ, Sae; REITZ, Emilie and BOCKELMAN, Patricia: Institutionalizing Blended Learning into Joint Training: A Case Study and Ten Recommendations (2014)
- [Fog09] FOGG, B. J.: A Behavior Model for Persuasive Design, in: *Proceedings of the 4th International Conference on Persuasive Technology*, Persuasive '09, ACM, pp. 1–7
- [Fow05] FOWLER, Martin: Event Sourcing (2005), URL <http://martinfowler.com/eaDev/EventSourcing.html>, last visited April 27, 2016
- [Fow11] FOWLER, Martin: CQRS (2011), URL <http://martinfowler.com/bliki/CQRS.html>, last visited April 27, 2016
- [Fri06] FRIEDENTHAL, Sanford; MOORE, Alan and STEINER, Rick: OMG Systems Modeling Language (OMG SysML) Tutorial, in: *INCOSE Intl. Symp*
- [Fül05] FÜLÖP, János: Introduction to Decision Making Methods, in: *BDEI-3 Workshop*
- [Gac15] GACKENHEIMER, Cory: *Introduction to React*, Apress, Berkely, CA, USA, 1st edn. (2015)

- [Gam95] GAMMA, Erich; HELM, Richard; JOHNSON, Ralph and VLISSIDES, John: *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Longman Publishing Co., Inc. (1995)
- [Gem13] GEMMA, Filippo: Future J-ISR (2013), URL <http://www.aofs.org/wp-content/uploads/2013/10/131010.08-Gemma-GMSpazio.pdf>, last visited April 27, 2016
- [Glo15] GLOBAL MILITARY COMMUNICATIONS: NCI Agency: Working MAJIIC (2015): pp. 16–18, URL <http://www.globalmilitarycommunications.com/wp-content/uploads/2016/01/NCI-Agency-working-MAJIIC1.pdf>
- [Goo13] GOODBOY DIGITAL LTD.: Pixi.js – 2D WebGL Renderer with Canvas Fallback (2013), URL <http://www.pixijs.com/>, last visited April 27, 2016
- [Gre09] GREGORY, Jason: *Game Engine Architecture*, Taylor & Francis Ltd. (2009)
- [Hai10] HAINEY, Thomas: Using Games-Based Learning to Teach Requirements Collection and Analysis at Tertiary Education Level (2010)
- [Har16] HARTMANN, Andreas: Web Components – Bereit für den Produktionseinsatz? (2016), URL <https://www.becompany.ch/de/blog/tech/2016/03/30/web-component.html>, last visited April 27, 2016
- [Hei02] HEIDER, Thomas and KIRSTE, Thomas: Supporting Goal-Based Interaction with Dynamic Intelligent Environments, in: *In Proc. 15th European Conference on Arti Intelligence (ECAI?2002*
- [Hei10] HEINZE, Norbert; ESSWEIN, Martin; KRÜGER, Wolfgang and SAUR, Günter: Image Exploitation Algorithms for Reconnaissance and Surveillance with UAV. *Airborne Intelligence, Surveillance, Reconnaissance (ISR) Systems and Applications VII* (2010)
- [Hun04] HUNICKE, Robin and CHAPMAN, Vernell: AI for dynamic difficulty adjustment in games. *Challenges in Game Artificial Intelligence AAAI* (2004): pp. 91–96
- [Hun13] HUNT, Pete: Why Did We Build React? (2013), URL <https://facebook.github.io/react/blog/2013/06/05/why-react.html>, last visited April 27, 2016
- [Hun14] HUNT, Pete: Rethinking Web App Development at Facebook (2014), URL <https://www.youtube.com/watch?v=nYkdrAPrdcw#t=32m>, last visited April 27, 2016
- [Hus09] HUSSAIN, Talib S.; ROBERTS, Bruce; MENAKER, Ellen S.; COLEMAN, Susan L.; POUNDS, Kelly; BOWERS, Clint; CANNON-BOWERS, Janis A.; MURPHY, Curtiss; KOENIG, Alan; WAINESS, Richard and LEE, John: Designing and Developing Effective Training Games for the US Navy. *Interservice/Industry Training, Simulation, and Education Conference* (2009)
- [Hwa81] HWANG, Ching-Lai and YOON, Kwangsun: *Multiple Attribute Decision Making: Methods and Applications A State-of-the-Art Survey*, Lecture Notes in Economics and Mathematical Systems, Springer Berlin Heidelberg (1981), URL <https://books.google.de/books?id=7yr6CAAAQBAJ>

- [Joh07] JOHNSON, W. Lewis; WANG, Ning and WU, Shumin: Experience with Serious Games for Learning Foreign Languages and Cultures, in: *Proceedings of the SimTecT Conference*
- [Kle07] KLEINBERGER, Thomas; BECKER, Martin; RAS, Eric; HOLZINGER, Andreas and MÜLLER, Paul: Ambient Intelligence in Assisted Living: Enable Elderly People to Handle Future Interfaces, in: *Universal access in human-computer interaction. Ambient interaction*, Springer (2007), pp. 103–112
- [Kno11] KNORR, René: Multi-Sensor-Verarbeitungssystem für Unmanned Aircraft Systems. *ESG Spektrum II/11* (2011), URL <https://www.esg.de/fileadmin/spektrum/ESG-Spektrum11-2.pdf>
- [Koc09] KOCH, Wolfgang; SIELEMANN, Marion and ULMKE, Martin: MAJIIC – ISR-Interoperabilität für weiträumige Bodenaufklärung, in: *Verteilte Führungsinformationssysteme*, Springer (2009), pp. 267–278
- [Kop16] KOPPERS, Tobias: Webpack Module Bundler (2016), URL <https://webpack.github.io/>, last visited April 27, 2016
- [Kor12] KORN, Oliver: Industrial Playgrounds: How Gamification Helps to Enrich Work for Elderly or Impaired Persons in Production, in: *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '12, ACM, pp. 313–316
- [Kor15] KORN, Oliver; FUNK, Markus and SCHMIDT, Albrecht: Towards a Gamification of Industrial Production: A Comparative Study in Sheltered Work Environments, in: *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, EICS '15, ACM, pp. 84–93
- [Li12] LI, Wei; GROSSMAN, Tovi and FITZMAURICE, George: GamiCAD: A Gamified Tutorial System for First Time AutoCAD Users. *UIST'12 - Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology* (2012): pp. 103–112
- [Lik32] LIKERT, Rensis: A Technique for the Measurement of Attitudes. *Archives of Psychology* (1932)
- [Mag09] MAGERKO, Brian: The Future of Digital Game-Based Learning. *Handbook of Research on Effective Electronic Gaming in Education* (2009): pp. 1274–1288
- [Mag14] MAGAÑA, Víctor Corcoba and ORGANERO, Mario Muñoz: The Impact of Using Gamification on the Eco-Driving Learning, in: *Ambient Intelligence-Software and Applications*, Springer (2014), pp. 45–52
- [Mar96] MARSHALL, Martin N.: Sampling for Qualitative Research. *Family Practice* (1996): pp. 522–525
- [Mar14] MARTIN, Matthew J.: Unifying Our Vision (2014): pp. 54–60, URL http://ndupress.ndu.edu/Portals/68/Documents/jfq/jfq-72/jfq-72_54-60_Martin.pdf
- [May02] MAYER, Richard E.; MATHIAS, Amanda and WETZELL, Karen: Fostering Understanding of Multimedia Messages Through Pre-Training: Evidence for a Two-Stage Theory of Mental

- Model Construction. *Journal of Experimental Psychology: Applied* (2002), vol. 8(3): pp. 147–154
- [McA14] MCANLIS, Colt; LUBBERS, Peter; JONES, Brandon; MAZUR, Andrzej; BENNETT, Sean; GARCIA, Bruno; LIN, Shun; POPELYSHEV, Ivan; HOWARD, Jon; BALLANTYNE, Ian and OTHERS: *HTML5 Game Development Insights*, Apress (2014)
- [Mic05] MICHAEL, David and CHEN, Sande: *Serious Games: Games that Educate, Train, and Inform*, Thomson Course Technology PTR (2005)
- [MM10] MESMER-MAGNUS, Jessica and VISWESVARAN, Chockalingam: The Role of Pre-Training Interventions in Learning: A Meta-Analysis and Integrative Review. *Human Resource Management Review* (2010), (4): pp. 261–282
- [Mun11] MUNTEAN, CI Cristina Ioana: Raising Engagement in E-Learning Through Gamification. *The 6th International Conference on Virtual Learning ICVL 2011* (2011): pp. 323–329
- [Naa94] NAAB, Kenneth N. and REICHART, Gunter: Driver Assistance Systems for Lateral and Longitudinal Vehicle Guidance, in: *International Symposium on Advanced Vehicle Control (1994: Tsukuba-shi, Japan). Proceedings of the International Symposium on Advanced Vehicle Control 1994*
- [NAT14] NATO: More Than Just Information Gathering – Giving Commanders the Edge (2014), URL http://www.nato.int/cps/en/natolive/news_110351.htm, last visited April 27, 2016
- [NAT16a] NATO: Joint Intelligence, Surveillance and Reconnaissance (2016), URL http://www.nato.int/cps/po/natohq/topics_111830.htm, last visited April 27, 2016
- [NAT16b] NATO COMMUNICATIONS AND INFORMATION AGENCY: Joint Intelligence, Surveillance and Reconnaissance (2016), URL <https://www.ncia.nato.int/Our-Work/Pages/Joint-Intelligence-Surveillance-and-Reconnaissance.aspx>, last visited April 27, 2016
- [Nik15] NIKOLOV, Stoyan: The Future of Fast and Easy UI-Design (2015), URL <http://www.makinggames.biz/features/the-future-of-fast-and-easy-ui-design,8524.html>, last visited April 27, 2016
- [Nod16] NODE.JS FOUNDATION: (2016), URL <https://nodejs.org/en/>
- [Nys14] NYSTROM, Robert: *Game Programming Patterns*, Genever Benning (2014), URL <http://gameprogrammingpatterns.com/>
- [Osm12] OSMANI, Addy: *Learning JavaScript Design Patterns*, O'Reilly Media, Inc. (2012)
- [Pho16] PHOTON STORM LTD.: Phaser – A Fast, Fun and Free Open Source HTML5 Game Framework (2016), URL <http://phaser.io/>, last visited April 27, 2016
- [Pre01a] PRENSKY, Marc: *Digital Game-Based Learning*, McGraw-Hill (2001), URL <https://books.google.de/books?id=XBwiAQAAIAAJ>

- [Pre01b] PRENSKY, Marc: Digital Natives, Digital Immigrants. *On the Horizon* (2001), vol. 9: pp. 1–6
- [Pre03] PRENSKY, Marc: Digital Game-Based Learning. *Computers in Entertainment* (2003), vol. 1(1)
- [Pre05] PRENSKY, Marc: "Engage Me or Enrage Me": What Today's Learners Demand. *Educause Review* (2005): pp. 60–65
- [Rao08] RAO, R. Venkata: *Decision Making in the Manufacturing Environment*, Springer-Verlag London (2008)
- [Rec06] RECH, Jörg; DECKER, Björn and RAS, Eric: Intelligente Assistenz in der Softwareentwicklung 2006: Zusammenfassung der Ergebnisse (2006), URL <http://publica.fraunhofer.de/documents/N-50981.html>
- [Ree13] REEVES, Byron and READ, J. Leighton: *Total Engagement: How Games and Virtual Worlds Are Changing the Way People Work and Businesses Compete*, Harvard Business Press (2013)
- [Rol13] ROLLER, Wolfgang; BERGER, Anton and SZENTES, Daniel: Technology Based Training for Radar Image Interpreters. *2013 6th International Conference on Recent Advances in Space Technologies (RAST)* (2013): pp. 1173–1177
- [Rom08] ROMAN, Paul A. and BROWN, Doug: Games – Just How Serious Are They? *The Interservice/Industry Training, Simulation & Education Conference (IITSEC)* (2008), (8013)
- [Sev12] SEVERANCE, Charles: JavaScript: Designing a Language in 10 Days (2012): pp. 7–8
- [She88] SHERIDAN, Thomas B.: Task Allocation and Supervisory Control. *Handbook of Human-Computer Interaction* (1988)
- [Spr13] SPRINGER, Sebastian: *Node.js: Das Umfassende Handbuch*, Galileo computing, Rheinwerk Verlag GmbH (2013), URL <https://books.google.de/books?id=UEoxmgEACAAJ>
- [Sta16] STACK OVERFLOW: Unikong (2016), URL <https://unikong.github.io>, last visited April 27, 2016
- [Str11] STREICHER, Alexander; DAMBIER, Natalie and ROLLER, Wolfgang: Semantic Search for Context-Aware Learning. *Proceedings of the 2011 7th International Conference on Next Generation Web Services Practices, NWeSP 2011* (2011): pp. 346–351
- [Str13] STREICHER, Alexander and SZENTES, Daniel: Mobile Assistenz in der Bildauswertung. 55. *Fachausschusssitzung Anthropotechnik "Ausbildung und Training in der Fahrzeug- und Prozessführung"* (2013): pp. 71–85, URL <http://publica.fraunhofer.de/dokumente/N-268953.html>
- [Str14] STREICHER, Alexander; SZENTES, Daniel and ROLLER, Wolfgang: Scenario Assistant for Complex System Configurations. *IADIS International Journal on Computer Science and Information Systems* (2014), vol. 9: pp. 38–52
- [Str15] STREICHER, Alexander and ROLLER, Wolfgang: Towards an Interoperable Adaptive Tutoring Agent for Simulations and Serious Games. *International Conference on Theory and Practice in Modern Computing, MCCSIS 2015* (2015): pp. 194–197

- [Str16] STREICHER, Alexander; SZENTES, Daniel and GUNDERMANN, Alexander: Generic Game-Based Exercise Trainers for Joint Training. *Submitted, to be published in an International Journal on Technology Based Learning and Serious Games* (2016)
- [Swe13a] SWERTZ, Christian; HEBERLE, Florian; STREICHER, Alexander; BOCK, Jürgen; SCHMÖLZ, Alexander; FORSTNER, Alexandra; HENNING, Peter; BARGEL, Bela-Andreas and ZANDER, Stefan: A Pedagogical Ontology as a Playground in Adaptive Elearning Environments. *Informatik 2013 Jahrestagung der Gesellschaft für Informatik e.V. (GI)* (2013), vol. 220: pp. 1955–1960, URL <http://publica.fraunhofer.de/documents/N-283340.html>
- [Swe13b] SWERTZ, Christian; SCHMÖLZ, Alexander; FORSTNER, Alexandra and STREICHER, Alexander: Adaptive Learning Environments as Serious Games. *Proceedings of the International Conference on Education and New Developments* (2013): pp. 175–180, URL <http://publica.fraunhofer.de/documents/N-283341.html>
- [Sze08] SZENTES, Daniel; BARGEL, Bela-Andreas; BERGER, Anton and ROLLER, Wolfgang: Computer-Supported Training for the Interpretation of Radar Images. *7th European Conference on Synthetic Aperture Radar* (2008)
- [Sze11] SZENTES, Daniel; BARGEL, Bela-Andreas and STREICHER, Alexander: Enhanced Value Benefit Analysis of Game Frameworks as a Tool for Digital Serious Game Development. *2011 7th International Conference on Next Generation Web Services Practices* (2011): pp. 352–356, URL <http://publica.fraunhofer.de/documents/N-234244.html>
- [Tai11] TAIVALSAARI, Antero and MIKKONEN, Tommi: The Web as an Application Platform: The Saga Continues. *Proceedings – 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011* (2011): pp. 170–174
- [Til16] TILDE INC.: Ember.js – A Framework for Creating Ambitious Web Applications (2016), URL <http://emberjs.com/>, last visited April 27, 2016
- [Tor16] TORO GAMES S.L.: Paint Online - A free draw, art and creativity game for kids (2016), URL <http://kidmons.com/game/paint-online>, last visited April 27, 2016
- [VE06] VAN ECK, Richard: Digital Game-Based Learning: It's Not Just the Digital Natives Who Are Restless *EDUCAUSE Review* (2006), vol. 41(2): pp. 16–30
- [W3C16] W3C: Web Components Current Status (2016), URL <https://www.w3.org/standards/techs/components>, last visited April 27, 2016
- [Wag11] WAGNER, Boris and PEINSIPP-BYMA, Elisabeth: Fused Generic Geospatial- and Meta-Information for Situation- and Crisis-Management Using OGC- and STANAG-Services. *International Society for Photogrammetry and Remote Sensing-ISPRS-: Gi4DM* (2011), URL <http://publica.fraunhofer.de/documents/N-185952.html>
- [Wan05] WANDKE, Hartmut: Assistance in Human-Machine Interaction: A Conceptual Framework and a Proposal for a Taxonomy. *Theoretical Issues in Ergonomics Science* (2005), vol. 6: pp. 129–155

List of Figures

1.1	Elements of the JISR cycle (Source: [NAT16b]).	2
1.2	Exemplary model of a system of systems in the military (Source: [Fri06]).	2
1.3	Screenshot of the game Tactical Iraqi (Source: [Hai10]).	3
1.4	Scope and outline of the thesis.	5
2.1	Comparison of two gamified production environments (Source: [Kor15]).	8
2.2	Screenshot of the gamified mission page of GamiCAD (Source: [Li12]).	9
2.3	Screenshots of the tutorial system TutorCAD (Source: [Li12]).	9
2.4	Android Client of SCENAS (Source: [Str14]).	10
2.5	Screenshot of Flooding Control Trainer (Source: [Hus09]).	11
2.6	Screenshot of a task evaluation in SAR-Tutor (Source: [Rol13]).	13
2.7	Different application scenarios of SAR-Tutor (Source: [Rol13]).	13
2.8	Screenshot of a CPR pre-trainer (Source: [Cre10]).	14
2.9	The three components of BLTS (Source: [Fau14]).	15
3.1	Classification of MADM methods (Source: [Hwa81]).	18
3.2	How assistance systems extend the capabilities of humans (Source: [Wan05]).	19
3.3	Linking of an assistance system and an e-learning application (Source: [Str11]).	20
3.4	Possible classification of DGBL and related concepts (Source: [Bre10]).	21
3.5	Flow zone: boundary between boredom and anxiety (Source: [Che07]).	22
3.6	Fogg Behavior Model (Source: [Fog09]).	23
3.7	Screenshots of two games realized with Phaser (Source: [Pho16]).	24
3.8	The overall bundling process of Webpack (Source: [Kop16]).	27
3.9	Component model and data flow in React (Source: koalite.com).	28
3.10	Comparison of React and the rendering engine of Doom 3 (Source: [Hun14]).	29
4.1	Criteria categories, subcategories and their weights.	34
5.1	Example process for image acquisition in a joint exercise (Source: [Str16]).	40
5.2	The modular game design concept of EXTRA (Source: [Str16]).	41
5.3	Screenshot of the EXTRA prototype.	42
5.4	Gameplay and game loop of EXTRA.	43
5.5	Architectural overview of EXTRA.	44
5.6	Client-Server communication in EXTRA.	44
5.7	Stages in the development of EXTRA.	47

5.8	Model of the level data structure in EXTRA.	48
5.9	Model of the view hierachy in EXTRA.	51
5.10	Screenshot of EXTRA highlighting the main React UI components.	52
6.1	Outline of the evaluation approach.	55
6.2	TCPED model which formed the basis of the scenario (Source: [Gem13]).	56
6.3	The JISR process used for the scenario and embedded into the game.	57
6.4	The three game levels of the scenario.	58
6.5	One of the participants playing EXTRA on an Apple iPad.	59
6.6	Recorded level completion times.	61
6.7	Results of the questions related to the game concept and usability.	61
6.8	Results of the test of knowledge.	62

Appendix A: Game Engine Analysis

1 Categories and Criteria

Criterion	Description	Weight	Aggregate
License	The license of the engine can increase the overall cost and complicate the licensing of the game.	3	
License Costs	A less expensive license reduces the total cost.	1	min
License Type	Permissive licenses do not pose limitations on the licensing of the game.	fatal (0 or 10)	min
Documentation	A good documentation and the availability of quality resources makes it easier for new developers to get started working with it.	6	
API Specification	A detailed and user-friendly API specification makes it easier to understand and write code, especially in a dynamic language such as JavaScript.	6	avg
Tutorials	Comprehensive tutorials help understand the concepts of the engine as well as common practices, but may only solve very specific problems and can be time-consuming to read.	4	avg
Examples	A thorough collection of examples can provide an overview of the features provided by the engine as well as inspiration and starting point for own implementations.	7	avg
Development	Criteria in this category make the game more accessible and ease development.	6	
Ecosystem	A broad ecosystem of plugins or components with a good discoverability can speed up development through code re-use.	8	avg
Debugging Utilities	Debugging utilities can help track bugs and other problems, and offer various insights into the application.	5	avg
API Simplicity	A clear, small and simple API eases development as less components and methods have to be considered, and less features are duplicated.	7	10 - min(10, 20 - a - b)

Criterion	Description	Weight	Aggregate
Community Support	A highly active community can help solving individual problems by providing direct support and by making it easier to find solutions to common problems via search engines.	7	avg
Supported Platforms	Supporting more devices and environments makes the game more accessible.	6	min
State/Stability	A mature and stable game engine should be more reliable and cover more essential features. Popular engines which are still being actively developed are less likely to be abandoned in the near future and more likely to implement new features as the underlying technology advances. These criteria contribute towards to the trust and longevity of the game engine.	5	
Corporate Backing	Sponsored projects are less likely to be abandoned due to a lack of resources, while also increasing the trust towards the engine, especially when backed by a major company in the field.	6	avg
Maturity	Engines that are still in early development are likely to be less reliable and have breaking API changes in the future.	8	avg
Activity	Inactive projects are likely to have bugs that will never be fixed. Active projects are more likely to add support for future technologies and adapt to changes.	6	avg
Popularity	Popular engines are less likely to be abandoned and more likely to contain essential features that have been overlooked.	4	avg
Unit Tests	Thoroughly tested code should make the engine more reliable and less prone to regression bugs, and allows testing locally in target environments/browsers.	5	avg
Features	Built-in support for the listed features eases the development of the game and are the very reason to use a game engine/UI library.	10	
User Interface	As a turn-based strategy game, the game requires a lot of UI elements.	8	max
TMX Support	TMX support is useful to allow creating levels with a visual editor (Tiled). This feature could also be realized with an additional parser library, however.	3	max
State Management	Top-level states are an useful feature to control and transition between different scenes (e.g. title screen, options, game).	5	max

Criterion	Description	Weight	Aggregate
Scene Graph	Scene Graphs are necessary to structure the rendering output of the game. Leaf nodes (entities or display objects), such as sprites, are used to visualize elementary game objects; inner nodes (containers) enable composition.	7	max
Client-Server Support	Client-Server architecture is planned, but can also be realized with a separate library for communication and additional design and implementation effort.	6	max
Animations	As in almost every video game, animations are very likely to be required.	6	avg
Physics	Physics are not a requirement, but could be useful in the future.	2	max
Audio	Sounds are essential to immersive games, but can also be realized with a separate library.	3	max

2 Evaluation Schemas

Criterion	0-3 Points	4-7 Points	8-10 Points
License			
License Costs	High licensing/royalty fees	Reasonable fees; simple pricing model	Low or no fees; simple pricing model
License Type	License prohibiting commercial use or infecting source code	-	Permissive license
Documentation			
API Specification	No online specification; only embedded in source code	Detailed auto-generated specification; some examples; user-friendly (clear structure, hyperlinks, search)	Detailed auto-generated or hand-written; many examples; very user-friendly
Tutorials	No or few tutorials	Few tutorials; tutorials are comprehensive and up-to-date	Many tutorials; list of community tutorials on website
Examples	No or few examples; only (partial) code examples	Some examples; focused feature examples as well as comprehensive ones	Many categorized examples; focused feature examples as well as comprehensive ones; live runnable examples

Criterion	0-3 Points	4-7 Points	8-10 Points
Development			
Ecosystem	No or few plugins	Some plugins; dedicated API	Many up-to-date plugins; dedicated API; API is well documented; web repository
Debugging Utilities	No / small official utilities or undocumented	Official utilities with few features or difficult to use	feature-rich official utilities; well documented; easy to use
API Simplicity	Many components with their own APIs and overlapping features; a small proportion of the API is expected to be used	Many methods, classes and/or properties (typically large library or framework) some or many of which are expected to be used	Few methods, classes and/or properties (typically library), many or all of which are expected to provide value
Community Support	Few support channels; low activity	Multiple support channels; medium activity	Multiple support channels; high activity; channels are listed on official website
Supported Platforms	Only specific browsers or recent versions are supported	Most popular browsers are supported; IE9+ is supported	Most popular browsers are supported; IE9+ is supported; mobile devices are supported
State/Stability			
Corporate Backing	Not created or sponsored by any company	Created or sponsored by a small company; official communication channels (blog, twitter)	Created or sponsored by major companies; official communication channels (blog, twitter)
Maturity	Still in early development; not considered stable or production-ready	Current release is considered stable / production-ready	Current release is considered stable / production-ready; first public release > 1 year ago
Activity	No updates in the last year	Few updates; mainly bugfixes	Several updates in the last year; bugfixes and features
Popularity	0-500 GitHub stars	500-10,000 GitHub stars	10,000 - 30,000 GitHub stars
Unit Tests	Non-existent or undocumented	Test files are easy to find; tests can be run with few commands	Test files are easy to find; tests can be run with a few commands; test coverage is measured and reasonably high; online CI server

Criterion	0-3 Points	4-7 Points	8-10 Points
Features			
User Interface	Built-in support only for basic elements like text; helpful features like hit-testing and input events; few resources	Some elements built-in; rich UI system as plugin; many resources	Rich UI system and many elements built-in; automatic layout management; well documented
TMX Support	Unsupported, experimental or lacking documentation	Basic support built-in or via plugin	Built-in support; well documented; feature-rich
State Management	Unsupported or experimental	Basic support built-in or via plugin; well documented	Built-in support; well documented; feature-rich; scenes/states can be (un-)serialized
Scene Graph	No built-in support for object/entity composition, sprite objects or shapes	Basic support with focus on games (positioning, transformations, sprite objects)	Feature-rich (e.g. support for various objects, optional seamless WebGL rendering)
Client-Server Support	No built-in support; few community resources	Experimental/inofficial support; documented	Built-in support; well documented; feature-rich
Animations	Few of: spritesheet animations, tweens with easing, global update loop, manual updating via entity positions	Many of: spritesheet animations, tweens with easing, global update loop, manual updating via entity positions	All of: spritesheet animations, tweens with easing, global update loop, manual updating via entity positions
Physics	No built-in support; simple integration of third-party systems	Simple system built-in; documentation on how to integrate third-party systems	Sophisticated or multiple systems built-in; well documented
Audio	Unsupported or experimental	Built-in API	Built-in API; well documented; feature-rich (preloading, fallback formats)

3 Evaluations (1/2)

Game Engines (1/2)			
Criterion	Pixi.js v3.0.9	Phaser v2.4.4	melonJS v3.0.0
License			
License Costs	Free of cost	Free of cost	Free of cost
License Type	MIT	MIT	MIT
Documentation			
API Specification	JSDoc with some details and examples; hyperlinks	JSDoc with details and link; few small examples	JSDoc with little details; links; examples; no search
Tutorials	Only getting started guide on website; many community tutorials	Some comprehensive tutorials; many community tutorials aggregated on official website	Two comprehensive official tutorials; some community tutorials
Examples	Many of live features examples	Many live features examples as well as game examples	Some feature and game examples inside the repository
Development			
Ecosystem	No dedicated API; few plugins on wiki	Plugin API; some free and premium plugins on website; some on npm; lack of documentation	Plugin API; very few official plugins; none on npm; lack of documentation
Debugging Utilities	No official debugging tools	Built-in API for debugging various components on screen; well documented	Official debug panel plugin; documented
API Simplicity	Comprehensive API focused on rendering	Pixi's API + many game-specific features	Comprehensive but clear API
Community Support	Active forum; less active reddit	Very active forum; twitter; IRC; reddit	Active forum; gitter chat
Supported Platforms	Most popular browsers including IE9+; touch events	Most popular browsers including IE9+; touch events	Most popular browsers including IE9+; touch events
State/Stability			
Corporate Backing	Created by goodboydigital (HTML5 games); official blog and twitter	Created by Photon Storm (HTML5 games); official blog, newsletter and twitter	No corporate backing; official blog

Game Engines (1/2)			
Criterion	Pixi.js v3.0.9	Phaser v2.4.4	melonJS v3.0.0
Maturity	1.0 Release in Apr 2013	1.0 Release in Sep 2013	1.0 Release in Apr 2014
Activity	3.0.9 @ Dec 2015; 3.0.0 @ Apr 2015; features and bugfixes	2.4.4 @ Oct 2015; 2.0.0 @ Mar 2014; features and bugfixes	3.0.0 @ Feb 2016; 2.1.4 @ Sep 2015; 2.0.0 @ Nov 2014; features and bugfixes
Popularity	1,900 GitHub stars	11,300 GitHub stars	1,500 GitHub stars
Unit Tests	Coverage unknown; CI server; run locally	None (planned)	Coverage unknown; CI server; run locally
Features			
User Interface	Hit-testing; input events; text elements; feature-rich plugins	Hit-testing; Text elements; Input events; Feature-rich plugins/toolkits; many resources	Hit-testing; input events; text rendering; few resources
TMX Support	Inofficial plugin; no isometric support	Built-in; well documented; no isometric support	Built-in; well documented; isometric support
State Management	Not supported	Built-in concept of states; well documented; official tutorial on implementing serialization	Built-in concept of states; well documented
Scene Graph	Supports containers, sprites, texts, and graphics objects for drawing primitives; WebGL rendering	Supports containers, sprites, texts, and graphics object for drawing primitives and optimized objects; WebGL rendering	Supports containers, sprites, texts, and renderables for drawing primitives; WebGL rendering
Client-Server Support	Not supported; some online resources	Not supported; some online resources	Not supported; some online resources
Animations	Simple spritesheet animations; manual updating	Spritesheet animations, tweens, update loop for manual changes	Spritesheet animations, tweens, update loop for manual changes
Physics	Simple integration of other systems; no official resources	Three systems built-in (Arcade, P2, Ninja)	Lightweight system built-in; documented
Audio	Not supported	Built-in; well documented; with debugging	Built-in; well documented

4 Evaluations (2/2)

Criterion	Game Engines (2/2)		UI Libraries	
	CreateJS v0.8.2	Crafty v0.7.1	React v0.14.7	Ember.js v2.4.1
License				
License Costs	Free of cost	Free of cost	Free of cost	Free of cost
License Type	MIT	MIT/GPL dual-licensed	BSD (3-clause)	MIT
Documentation				
API Specification	JSDoc with details, links, search and many examples	Auto-generated with some details, links and many examples	Hand-written with many details and some examples	YUIDoc with many details, examples, guides and links
Tutorials	Some official tutorials; many community tutorials; not all focused on games	Short getting started and specific feature-oriented guides; some community tutorials	Many official guides; many community tutorials, some on games	Many official and structured guides and tutorials; many community tutorials, some outdated
Examples	Some feature and game examples inside the repository	Official examples as part of specification and guides; some live examples	Official examples as part of specification and guides	Official examples as part of specification and guides (code snippets)
Development				
Ecosystem	None	Many components listed on official website; some outdated	API is built around reusable components; unofficial repository; many components on npm	API is built around reusable components; unofficial repository; plenty of components on npm
Debugging Utilities	No official debugging tools	Limited to visualizing shapes related to entities (hitboxes, bounding rectangles)	Debug/Development mode for type-checking and developer warnings; official Chrome extension	Opt-in logging; official browser extensions
API Simplicity	Split up into four libraries; clear rendering API	comprehensive API	Small and clear API; HTML & CSS required	Comprehensive API; HTML & CSS required

Criterion	Game Engines (2/2)		UI Libraries	
	CreateJS v0.8.2	Crafty v0.7.1	React v0.14.7	Ember.js v2.4.1
Community Support	Reddit; google plus community; medium activity	Less active google groups from linked from official website	Forum, IRC, twitter listed on official website; reddit; very high activity	StackOverflow, IRC, forum linked from official website; reddit; very high activity
Supported Platforms	Most popular browsers including IE9+; touch events	Most popular browsers including IE9+; touch events	Most popular browsers including IE9+; support for touch events and responsive layout (CSS)	Most popular browsers including IE9+; support for touch events and responsive layout (CSS)
State/Stability				
Corporate Backing	Created by gskinner; official twitter and blog; sponsored by Adobe, Microsoft, AOL and Mozilla	Not created or sponsored by any company	Created by facebook; official blog and twitter account	Some core team members belong to tilde inc; blog; twitter, Google Plus; sponsored by LinkedIn and Yahoo
Maturity	0.3 Release in Feb 2011	0.1 Release in Dec 2010	Released in May 2013; production-ready	Forked from SproutCore in Dec 2011
Activity	0.8.2 @ Nov 2015; 0.8.0 @ Dec 2014; features and bugfixes	0.7.1 @ Feb 2016; 0.7.0 @ Nov 2015; 0.6.0 @ Dec 2013; features and bugfixes	0.14.7 @ Jan 2016; 0.14.0 @ Oct 2015; features and bugfixes	2.4.1 @ Mar 2016; 2.0.0 @ Aug 2015; features and bugfixes
Popularity	5,000 GitHub stars (EaselJS) + 5,200 GitHub stars (SoundJS + TweenJS + PreloadJS)	2,000 GitHub stars	36,100 GitHub stars	16,000 GitHub stars
Unit Tests	Coverage unknown; run locally	Coverage unknown; CI server; run locally	87% Coverage; CI server; run locally	Coverage unknown; CI and code quality / health server; run locally

Criterion	Game Engines (2/2)		UI Libraries	
	CreateJS v0.8.2	Crafty v0.7.1	React v0.14.7	Ember.js v2.4.1
Features				
User Interface	Hit-testing; text elements; input events	Hit-testing, Text rendering, Input Events, HTML entities	Built for designing complex HTML & CSS Uis	Built for designing complex HTML & CSS Uis
TMX Support	Not supported	Via unofficial plugin which was last updated > 1 year ago	Not supported	Not supported
State Management	Not supported	Built-in concept of scenes	Not supported; can be implemented as components	Not supported; can be implemented as components
Scene Graph	Supports containers, sprites, and graphics for drawing primitives and optimized objects; WebGL in preview state	Supports entities that can also act as containers, e.g. sprites, texts, images and canvas for drawing primitives; not all objects support WebGL rendering	Scene Graphs can be implemented as components; sprites and other objects via HTML & CSS	Scene Graphs can be implemented as components; sprites and other objects via HTML & CSS
Client-Server Support	Not supported; few online resources	Experimental, unofficial plugin	Not supported	Not supported
Animations	Spritesheet animations, tweens, update loop for manual changes	Spritesheet animations, tweens, update loop for manual changes	Spritesheet animations and tweens via CSS	Spritesheet animations and tweens via CSS
Physics	Simple integration of other systems; no official resources	Gravity and collision detection built-in; no collision handling; community efforts to integrate physics engines	Not supported	Not supported
Audio	Built-in; well documented	Built-in; few features	Not supported	Not supported

5 Ratings

	Game Engines					UI		Combinations									
	Pixi.js	Phaser	melonJS	CreateJS	Crafty	React	Ember.js	Pixi.js + React	Pixi.js + Ember.js	Phaser + React	Phaser + Ember.js	melonJS + React	melonJS + Ember.js	CreateJS + React	CreateJS + Ember.js	Crafty + React	Crafty + Ember.js
License Costs	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
License Type	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
∑ License	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
API Specification	7	4	4	8	4	9	10	8	8.5	6.5	7	6.5	7	8.5	9	6.5	7
Tutorials	3	9	5	2	4	9	8	6	5.5	9	8.5	7	6.5	5.5	5	6.5	6
Examples	5	10	4	4	7	5	3	5	4	7.5	6.5	4.5	3.5	4.5	3.5	6	5
∑ Documentation	5.2	7.6	4.2	4.9	5.2	7.4	6.6	6.3	5.9	7.5	7.1	5.8	5.4	6.1	5.8	6.3	5.9
Ecosystem	2	7	3	0	3	10	10	6	6	8.5	8.5	6.5	6.5	5	5	6.5	6.5
Debugging Utilities	0	7	5	0	2	9	10	4.5	5	8	8.5	7	7.5	4.5	5	5.5	6
API Simplicity	7	4	6	7	5	7	5	4	2	1	0	3	1	4	2	2	0
Community Support	5	10	4	6	3	10	10	7.5	7.5	10	10	7	7	8	8	6.5	6.5
Supported Platforms	8	8	8	8	8	10	10	8	8	8	8	8	8	8	8	8	8
∑ Development	4.5	7.2	5.1	4.2	4.2	9.2	8.9	6.0	5.7	7.1	6.9	6.2	5.9	5.9	5.5	5.7	5.3
Corporate Backing	6	7	2	10	0	10	10	8	8	8.5	8.5	6	6	10	10	5	5
Maturity	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10	10
Activity	10	10	10	6	8	10	10	10	10	10	10	10	10	8	8	9	9
Popularity	5	8	5	6	5	10	8	7.5	6.5	9	8	7.5	6.5	8	7	7.5	6.5
Unit Tests	7	0	7	6	7	10	8	8.5	7.5	5	4	8.5	7.5	8	7	8.5	7.5
∑ State/Stability	8.0	7.4	7.1	7.9	6.3	10	9.4	9.0	8.7	8.7	8.4	8.6	8.3	9.0	8.7	8.2	7.8
User Interface	5	6	4	3	2	10	10	10	10	10	10	10	10	10	10	10	10
TMX Support	4	7	9	0	3	0	0	4	4	7	7	9	9	0	0	3	3
State Management	0	8	7	0	6	3	3	3	3	8	8	7	7	3	3	6	6
Scene Graph	9	10	10	8	9	3	3	9	9	10	10	10	10	8	8	9	9
Client-Server	2	3	2	1	4	0	0	2	2	3	3	2	2	1	1	4	4
Animations	2	10	10	10	10	3	3	2.5	2.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5	6.5
Physics	2	9	5	2	3	0	0	2	2	9	9	5	5	2	2	3	3
Audio	0	10	9	9	6	0	0	0	0	10	10	9	9	9	9	6	6
∑ Features	3.6	7.6	6.8	4.4	5.7	3.4	3.4	5.0	5.0	7.9	7.9	7.5	7.5	5.7	5.7	6.7	6.7
∑ All Categories	5.5	7.7	6.3	5.6	5.8	7.1	6.8	6.6	6.4	8.0	7.8	7.3	7.1	6.8	6.6	7.0	6.8

Appendix B: Evaluation Documents

1 Introduction Document

Please See Next Page.

Exercise Trainer

In dem Spiel Exercise Trainer (EXTRA) geht es darum, verschiedene Prozess- bzw. Logistikketten aufzubauen, um Produkte zu erstellen, weiterzuverarbeiten, zu sammeln und an Kunden zu verkaufen. Hierzu müssen Fabriken und Märkte gebaut, und über Wegstrecken verbunden werden. Außerdem müssen unter Umständen die aktiven Prozesse in den Fabriken eingestellt werden, falls mehrere unterstützt werden.

Die genannten Gebäude, Prozesse, Wege und Kunden sollen dabei Elemente der realen Welt abstrahieren, z.B.:

- Fabriken: (Technische) Systeme, die Daten produzieren oder weiterverarbeiten, wie etwa Drohnen mitsamt Bodenstationen zur Fernsteuerung
- Märkte: Zielsysteme bzw. Anforderungsplätze
- Wege: Kommunikationskanäle wie etwa LAN, WLAN
- Produkte: Verschiedene Dokumente bzw. Informationen, z.B. aufgenommene Bilder
- Prozesse: Mögliche Aktivitäten einer Fabrik mit unterschiedlichen eingehenden bzw. ausgehenden Produkten, wobei nicht immer ein eingehendes Produkt gefordert wird, wie z.B. bei Bildaufnahme
- Kunden: Endnutzer, die Informationen anfordern und bedient werden müssen

Damit soll in erster Linie den Beteiligten von militärischen Großübungen mit komplexen Systemverbänden geholfen werden, sich gezielt vorbereiten zu können, indem sie die einzelnen Systeme und deren Aufgaben spielerisch und auf abstrakter Ebene kennenlernen.



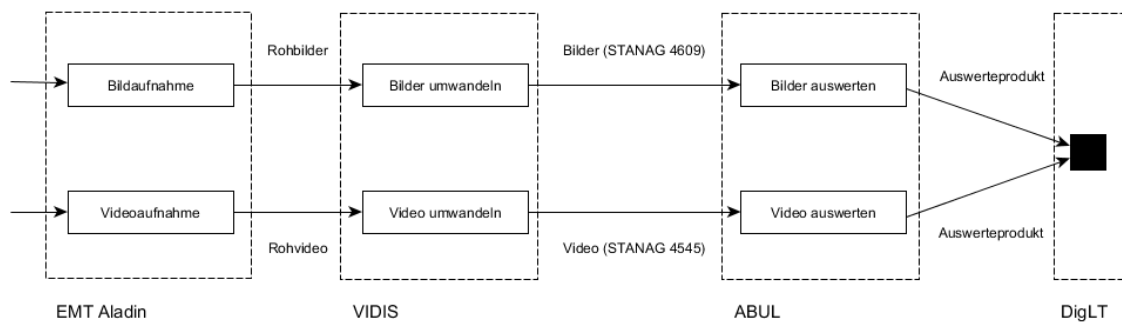
Komplexe Systemverbände bei militärischen Großübungen

Szenario

Zur Evaluation des EXTRA-Prototyps wurden insgesamt drei Spiel-Levels erstellt, basierend auf einem einfachen, imaginären Übungsszenario der NATO names „Spectral Tiger“. Teil des Szenarios ist ein Systemverbund zur Aufnahme und Auswertung von Bild- und Videodaten.

Stellen Sie sich vor, Sie seien nun Teilnehmer dieser Übung und sind als Operateur des Drohnensystems EMT Aladin für deren Einsatz verantwortlich. Da Sie die restlichen Systeme nicht kennen, möchten Sie nun im Vorfeld verstehen, welche Rolle EMT Aladin im Gesamtkontext einnehmen wird und wie die erzeugten Daten weiterverarbeitet bzw. ausgewertet werden sollen.

Das Gesamtsystem ist wie folgt aufgebaut:



EMT Aladin: Aufklärungsdrohne der Firma EMT

VIDIS: Video Distribution System; verteilt u.a. Videos und Bilder und unterstützt Weitergabe als Standard-Formate (STANAG)

ABUL: Automatisierte Bildauswertung für Unbemannte Luftfahrzeuge

DigLT: Digitaler Lagetisch; ein Multi-Display-Arbeitsplatz für verschiedene Analysen im Team

STANAG: Standardization Agreements (Standard-Richtlinien) der NATO-Staaten; hier also standardisierte Bild-/Videoformate

Ihre Aufgabe ist es nun, in den einzelnen Spiel-Levels die richtigen Produktionsketten aufzubauen, um die von den Kunden geforderten Produkte zu sammeln und zu verkaufen. Hierzu müssen Sie bei Bedarf die vorplatzierten Gebäude ergänzen, in geeigneter Weise einstellen und verbinden (entsprechend der ein- und ausgehenden Produkte), und schließlich die gesammelten Produkte verkaufen.

Da während der Levels unter Umständen verschiedene Produkte gefordert werden, und diese nicht direkt ersichtlich sind, müssen Sie außerdem stets auf die Kundenanfragen achten und ggf. die Produktionsketten anpassen bzw. erweitern oder weitere errichten. Brauchen Sie dafür zu lange, so bekommen Sie weniger Punkte beim Verkauf und verlieren allmählich an Reputation. Sinkt Ihre Reputation auf 0 oder weniger, so verlieren Sie das Level. Sobald alle Kunden zufrieden gestellt sind, ist das Level gewonnen.

2 Questionnaire

Please See Next Page.

Extra Evaluation

Diese Umfrage enthält 8 Fragen.

Spielkonzept

[]Bewertung *

Bitte wähle die zutreffende Antwort aus:

	1: Stimme überhaupt nicht zu	2	3	4	5: Stimme vollkommen zu
Mir gefällt das Spielkonzept	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Das Konzept entspricht dem zuvor gezeigten Einführungsszenario	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Die Schwierigkeit des Spiels war angemessen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Die Spiellänge war angemessen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

[]Haben Sie weitere Anmerkungen zum Spielkonzept?

Bitte gib hier Deine Antwort ein:

Benutzerfreundlichkeit

[]Bewertung *

Bitte wähle die zutreffende Antwort aus:

	1: Stimme überhaupt nicht zu	2	3	4	5: Stimme vollkommen zu
Ich denke, ich würde das Spiel öfters spielen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Ich fand das Spiel einfach zu bedienen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Ich denke, die meisten Leute würden schnell lernen, das Spiel zu bedienen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Ich musste eine Menge Dinge lernen, bevor ich anfangen konnte, zu spielen	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

[]Haben Sie weitere Anmerkungen zur Benutzerfreundlichkeit?

Bitte gib hier Deine Antwort ein:

Wissensfragen

[] *

Bitte wähle die zutreffende Antwort aus:

	Ja	Unsicher	Nein
Das System CSD war Teil des vorgestellten Systemverbunds.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Das System DigLT war Teil des vorgestellten Verbunds.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Das Drohnensystem EMT Luna war Teil des vorgestellten Verbunds.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Das System VIDIS diente zur Umwandlung von Rohvideos und -bildern in (STANAG) Standardformate.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Gesamtziel des Szenarios war die Erstellung von Bildauswerteprodukten.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Mit dem System ABUL ließen sich die mit EMT Aladin produzierten Bilder und Videos direkt zu Auswerteprodukten weiterverarbeiten.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Das System ABUL war Teil des vorgestellten Verbunds.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
VIDIS wird hauptsächlich dazu benutzt, Videos und Bilder aufzunehmen.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Demografische Daten

[]Wie alt sind Sie? *

Deine Antwort muss zwischen 14 und 99 liegen.
In diesem Feld kann nur ein ganzzahliger Wert eingetragen werden.

Bitte gib hier Deine Antwort ein:

Jahre

[]Sind Sie männlich oder weiblich? *

Bitte wähle nur eine der folgenden Antworten aus:

- Weiblich
 Männlich

[]Was ist Ihr höchster allgemeinbildender Schulabschluss? *

Bitte wähle nur eine der folgenden Antworten aus:

- Kein Abschluss
 Hauptschulabschluss
 Realschulabschluss
 Fachhochschul- oder Hochschulreife
 Sonstiges

Danksagung

Die vorliegende Masterarbeit entstand im Rahmen meines Studiums am KIT in Karlsruhe und in Zusammenarbeit mit dem Fraunhofer IOSB. In diesem Zusammenhang möchte ich mich gerne bei Alexander Streicher für die großartige Betreuung und Unterstützung bedanken.

Außerdem bedanke ich mich bei allen Teilnehmern der Nutzerstudie, die sich kurzfristig und trotz mangelnder Entschädigung zur Teilnahme bereit erklärt haben.

Des Weiteren möchte ich mich bei Sabrina Bauer für das Korrekturlesen dieser Arbeit und weitergehende Ratschläge bedanken.

Abschließend danke ich besonders meinen Eltern Inge und Helmut Gundermann, die mich während meines gesamten Studiums moralisch und finanziell unterstützt haben.