Automated Scenario Generation

Coupling Planning Techniques with Smart Objects

Technical Artificial Intelligence

MSc Thesis

Author:
Gwen R. Ferdinandus

Supervisors:
MSc. Marieke Peeters
Dr. Karel van den Bosch
Prof.Dr. John-Jules Ch. Meyer

October 1, 2012

Abstract

To facilitate autonomous training in virtual environments, recent studies have explored the integration of didactically founded methods, such as Scenario-Based Training (SBT), into serious games. The resulting Adaptive Educational Games (AEGs) monitor the performance of the trainee, and adapt the game to match this performance. To ensure effective training opportunities, an AEG should not only adapt the game during the scenario, but should also select appropriate scenarios based on the needs and abilities of the trainee. However, manual scenario creation is a time-consuming and often costly process. Writing personalised scenarios on a larger scale is simply not feasible. Consequently, the need arises for automated scenario generation techniques. In this thesis I present a framework for an automated scenario generation system. First, I consider the requirements of effective training scenarios. Next, the design, implementation, and evaluation of an automated scenario generator are discussed. The design of the framework is based on the idea that the content of the game, combined with the actions of the virtual agents, determines the training experience for the trainee. Therefore, the framework couples the planning of trainee actions with the selection of appropriate content for the virtual environment. To test the framework, a prototype has been implemented for the domain of burn-related incidents in First Aid training. The scenarios generated by this prototype have been compared to those written by human experts and laymen. The results followed the hypothesised trend with the expert scenario being judged best, followed by those generated by the sytem, and finally the laymen scenarios. However, all differences were not significant. It seems likely that there will be significant differences, at least between the expert and layman scenarios, in experiments with higher power. Further research will be required before conclusive answers can be given. For now, the combination of automated planning techiques with Smart Objects seems a viable approach to automated scenario generation. At the very least, the framework warrants further research, and several directions for additional studies have been identified. Adding an automated scenario generator to an AEG will provide the game with even more personalisation capabilities, and offer the trainee varied and appropriate training experiences.

Acknowledgements

I would like to start by thanking all the people who have supported me during my first major academic trial. First of all my supervisors: Karel van den Bosch, and John-Jules Meyer for their continued support and constructive critiscm, and especially Marieke Peeters for her dedication, motivation, and most importantly for asking how I was doing before asking how my thesis was going. I would also like to thank Jan Broersen for helping me find this research opportunity at TNO.

Furthermore, I wish to express my sincere gratitude to the First Aid instructors, laymen, and military officers who have so graciously invested their time and expertise to help me with my research.

A different kind of thanks goes out to my family and friends who were always there for me. Thank you, Armon en Shoshannah for your distractions, your support, and your continued friendship. Patrick, to you my most heartfelt thanks, for your love and unwavering support, even after sixty pages of spell-checking.

Contents

1 Introduction **1**
- 1.1 Problem Description . . . 1
- 1.2 My Focus . . . 1
- 1.3 Thesis overview . . . 2

2 Problem Domain Analysis **3**
- 2.1 SBT in literature . . . 3
- 2.2 SBT in Practice . . . 5
- 2.3 Scenario Requirements . . . 7

3 Background **9**
- 3.1 Automated Planning . . . 9
  - 3.1.1 Conceptual Model for Planning . . . 9
  - 3.1.2 Classical Planning . . . 11
  - 3.1.3 Types of planners . . . 12
  - 3.1.4 HTN planning . . . 14
  - 3.1.5 Hybrid planning . . . 14
- 3.2 Smart Objects . . . 16

4 Scenario Generation Framework **19**
- 4.1 Scenario Generation for Adaptive Educational Games . . . 19
- 4.2 Overview of the Scenario Generator System . . . 20
- 4.3 Action Planner . . . 21
  - 4.3.1 Planning Domain . . . 22
  - 4.3.2 Planning Process . . . 23
- 4.4 Object Selector . . . 25
  - 4.4.1 Smart Objects . . . 26
  - 4.4.2 Selection Process . . . 26
- 4.5 Scenario Requirements . . . 27

5 Implementation **29**
- 5.1 Training Domain: First Aid . . . 29
- 5.2 Prototype . . . 29
  - 5.2.1 Input . . . 30
  - 5.2.2 Output . . . 32
  - 5.2.3 General Implementation Considerations . . . 32
  - 5.2.4 Scenario Generation Process . . . 33

6 Evaluation **35**
6.1 Introduction . . . 35
6.1.1 Research Questions and Hypotheses . . . 36
6.1.2 Domain Choice . . . 36
6.2 Method . . . 37
6.2.1 Participants . . . 37
6.2.2 Research Design . . . 37
6.2.3 Materials . . . 37
6.2.4 Measures . . . 39
6.2.5 Procedure . . . 40
6.2.6 Analysis . . . 40
6.3 Results . . . 41
6.3.1 Data Exploration . . . 41
6.3.2 Inter-rater Reliability . . . 42
6.3.3 Missing values . . . 42
6.3.4 Repeated Measure Analysis . . . 42
6.4 Discussion of the Results . . . 43

7 Discussion **45**
7.1 Related Work . . . 45
7.2 Direction for Further Research . . . 47
7.2.1 Extendibility of the Prototype to Other Training Domains . . . 47
7.2.2 Task and Competency Suitability . . . 49
7.2.3 Variety . . . 49
7.2.4 Comparison . . . 50

8 Conclusion **51**

Chapter 1

Introduction

Over the course of the last two decades, serious games have increased in popularity as an educational tool. Improvements in graphics hardware and software have enabled the creation of realistic virtual environments where trainees can practise tasks that would be too dangerous or costly in real life. With the introduction of intelligent agents to control key characters during the game, trainees now have the opportunity to practise without the need for team members or instructors to be present ([45]). This kind of autonomous training sessions can greatly reduce the costs of, and increase the opportunity for, practice. However, the use of a virtual environment by itself is not enough to ensure effective training. Without appropriate guidance, "negative" training can occur leading to misconceptions in the trainee's understanding ([24]). To overcome this problem and ensure effective training, serious games have to incorporate didactically founded guiding mechanisms for the trainee.

1.1 Problem Description

The need for guidance of the trainee has inspired researchers to examine the possibilities of integrating didactic principles and proven training methods into serious games (see for example [37], [38], [43]) The resulting Adaptive Educational Games (AEGs) provide structure to the training, and ensure the occurrence of appropriate training experiences. They monitor the trainee's performance and adapt the game to ensure effective training. One training methodology that lends itself especially well for integration into AEGs is Scenario Based Training (SBT). SBT exemplifies the learning-by-doing approach to training. In SBT, trainees practise complex tasks in controlled environments representative of the actual working domain. The trainee experiences a simulated course of events (the scenario) that is purposefully designed to offer the desired training opportunities. Originally limited to military use, SBT is now widely accepted and applied extensively in a large variety of domains such as aviation, medicine, and negotiation. The story-like nature of SBT makes it a natural candidate to provide structure and guidance for trainees in AEGs.

Integrating SBT principles into AEGs offers the didactical structure needed to provide appropriate training experiences for the trainee. However, manual scenario creation is a time-consuming process, often requiring input from both game programmers and domain experts. As a result, most systems reuse a limited set of scenarios. Although this approach can work for new trainees, it does not facilitate continued training. Moreover, to ensure effective training, scenarios should be adapted to the needs and abilities of the individual trainee, and offer him varied situations to practise with. As such, the need for automation of the scenario generation process arises.

1.2 My Focus

To fulfil the growing need for large numbers of personalised training scenarios, I propose a framework for automated scenario generation. An automatic scenario generator is a computational system that produces personalised training scenarios, given knowledge about the learning objectives, the needs and abilities of the trainee, and the training domain. A scenario is defined as an intial world state, and a sequence of events that is expected to follow from this state. The framework I propose is inspired by the work of Lopes and Bidarra (2011) [27]. They state that *"the fulfilment of a game scenario within the game world defines and characterizes the majority of the player experience,"* and argue for the integration of scenario and content generation techniques to offer trainees individualised experiences. Following this concept, the proposed framework combines an automated scenario planner with a content selection mechanism.

The automated scenario planner is inspired by the work of Niehaus and Riedl (2009) on automated scenario adaptation [34]. They introduce a mechanism to adjust predefined scenario plans to the needs and abilities of a specific trainee. A predefined plan consists of a sequence of abstract high-level tasks that the trainee has to perform. Based on a model of the trainee's skills, this predefined plan is adapted by removing, adding and/or changing task, so as to better fit his needs. An automated planning algorithm is then employed to ensure the consistency between the tasks and decompose the selected high-level tasks into concrete actions. Together these actions constitute an action plan for the trainee. This approach allows for a select number of predefined scenario plans to suffice in facilitating a large number of trainees.

The task of the content selection mechanism is to instantiate the scenario plan in the virtual world, and prompt the desired behaviour of the trainee by careful selection of the appropriate game world content. For a system to reason about appropriate content, it must have knowledge concerning the interaction possibilities each object offers. Therefore, the framework builds on the notion of Smart Objects as introduced by Kallmann and Thalmann (1998) [21]. These Smart Objects are annotated with additional semantic information regarding their interaction possibilities, and associated agent behaviour.

Combining an automated action planner with a Smart Object selection mechanism creates a unique system that can generate varied training scenarios tailored to the needs and abilities of the individual trainee. In this thesis, I present the design of this framework and evaluate the performance of a prototype to address the question: "Can a system automatically generate effective training scenarios for AEGs by coupling automated planning techniques with the use of Smart Objects?"

1.3 Thesis overview

The next chapter covers the theory and practice behind SBT to determine the requirements of effective training scenarios. Chapter 3 lays the theoretical foundations for my work, and provides a general introduction into the fields of automated planning and Smart Objects. Chapter 4 then presents the proposed framework for automatic scenario generation, explaining the various components and their interaction in detail. Chapter 5 outlines the choices made and the challenges encountered during the implementation of the prototype. The experiment conducted to evaluate the performance of this prototype is described in Chapter 6. This chapter also discusses the results and suggests points for improvement regarding the experimental design. Chapter 7 places the framework in context with existing research into automated scenario generation and discusses directions for future research. Finally, Chapter 8 lists the conclusions that can be drawn from this research project.

Chapter 2

Problem Domain Analysis

The problem addressed in this project is how to automatically generate effective scenarios for use in serious games. An important question that must be answered is: what constitutes an effective scenario? To identify these requirements, this chapter analyses the problem domain discussing the theory and practice of Scenario-Based Training (SBT). The first section presents the arguments gathered from literature studies, and the second section reports on interviews conducted with several instructors experienced in the use of SBT. The third and final section provides a brief summary, combining the requirements.

2.1 SBT in literature

SBT is a training method that focusses on systematic practice and feedback in realistic situations. It is often applied in military training, but can also be recognised in the role play sessions in medical and soft skill domains. An SBT session is based on an a priori defined scenario. Such a scenario consists of a set of events that are to trigger specific behaviours in the trainee. Scenarios for SBT have to be carefully designed such that opportunities for the trainee to practise and demonstrate his proficiency regarding the targeted skills are not left to chance. SBT design methods emphasize the necessity of the explicit linkage between training objectives, scenario design, and observation and feedback. It is because of this linkage that SBT can ensure that learning occurs ([12], [36]). When designing a scenario, events are systematically introduced to create the requirement for the trainee to act. This structured design approach makes SBT diagnostic, meaning that it facilitates observation and assessment of the trainee. Observers can anticipate the trainee's behaviour, they know what to look for and when. Figure 2.1 shows the major components of SBT. A quick overview of the different steps will be given below. The interested reader is referred to [36] and [40] for a more elaborate description including examples, and to [3] for additional analysis of the challenges associated with each step.

The cycle starts with the identification of the knowledge, skills, and attitudes underlying effective task performance. This information can be obtained from task lists, task analysis or subject matter experts. The second step is to select learning objectives for the current training based on the trainee's mastery of the required competencies. These objectives have to be clearly defined and quantifiable and can refer to task-specific or more general competencies. In the next step, events that provide the trainee with opportunities to practise each of the targeted competencies are combined into a scenario. These events have to be structured in such a way that the natural flow and believability of the scenario are maintained. Step four in the SBT cycle is the development of performance measures that will diagnose the progress of the trainee. These measures should be tightly linked to the scenario events and the associated competencies. Effective performance measures should not only register if the trainee is improving, but also why he is behaving as observed. In performance diagnosis, the behaviour of the trainee is analysed to discover misconceptions, knowledge gaps or lacking skills that need to be addressed. This step requires the mapping of observed behaviour to the underlying competencies. Step six then deals with providing feedback to the trainee. Feedback is of paramount importance to the learning process. Because of the linkage between training objectives, scenario

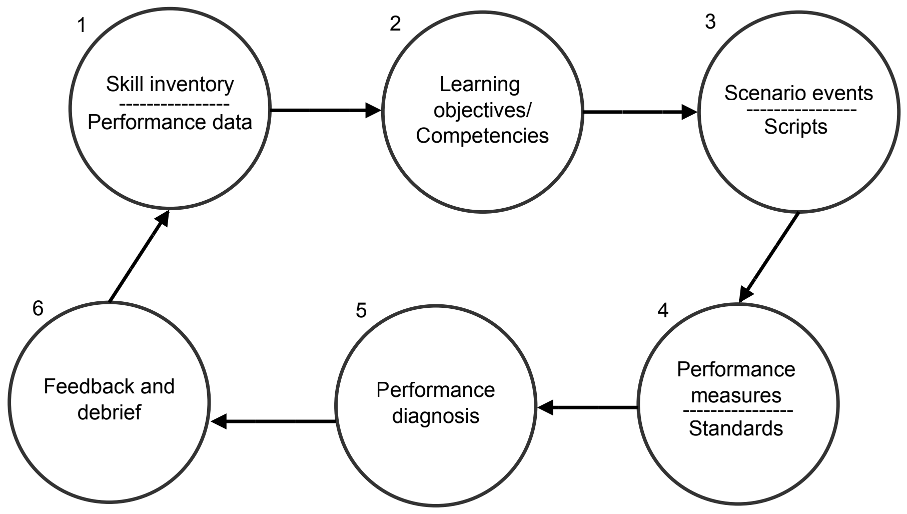

Figure 2.1: Main components of SBT, copied from [6]

events and performance measures, SBT allows the instructor to give very specific feedback with respect to the targeted skills. To close the loop, the performance data is used to update the skill inventory of the trainee and ensure that future sessions build on what has already been learned.

The general process underlying SBT reveals a need for structured guidance in designing training scenarios. Learning objectives have to be carefully selected to ensure that the scenario design is firmly grounded and didactically structured. These objectives then directly determine the events that constitute the scenario and afford the trainee with practice opportunities. Several recent studies have identified additional design requirements for effective training scenarios ([28], [37], [50]). In the following I will focus on the work of Peeters et al. (2012) [37], because it encompasses the others and provides the most extensive explanation. Using the situated Cognitive Engineering method Peeters et al. have analysed operational demands for an AEG employing SBT. They list four requirements of scenario design. First of all, the tasks offered to the trainee should be appropriate for his current skill level. Secondly the situations should resemble real-world situations that the trainee may encounter during his future job performance. Moreover the scenarios should portray a realistic course of events. Finally, variable scenarios have to be generated. Below I briefly elaborate on these requirements and the claims they are based on.

Offering tasks appropriate to the competency level of the trainee is important for effective learning. This claim is based on two well-known and similar concepts; the Zone of Proximal Development (ZPD) of Vygotsky ([46]) that is commonly used in instructional theories, and Csikszentmihalyi's Flow ([8]), a concept generally associated with game design. The ZPD represents a class of tasks that challenge the trainee because they ask slightly more than his current skill level can offer, yet they do not overwhelm him. With appropriate support the trainee will be able to perform the tasks within the ZPD and in doing so he will develop new skills and insights. Tasks outside the ZPD are either too difficult, confusing the trainee, or too boring. The concept of Flow involves a slightly different perspective and focuses on the mental state that can be achieved by properly matching challenges and skills. When a player is "in the flow" he is usually highly motivated and learning efficiently. Figure 2.2 shows two graphs representing the ZPD and Flow. By offering tasks appropriate to the competency level of the trainee, a scenario adheres to the principles of the ZPD and Flow. To accomplish this match, a scenario designer requires knowledge on both the skill level of the trainee and the complexity level of the tasks. Extensive research has been performed regarding Student Models that automatically keep track of the knowledge and skills of a trainee, a few examples are [16], [23]

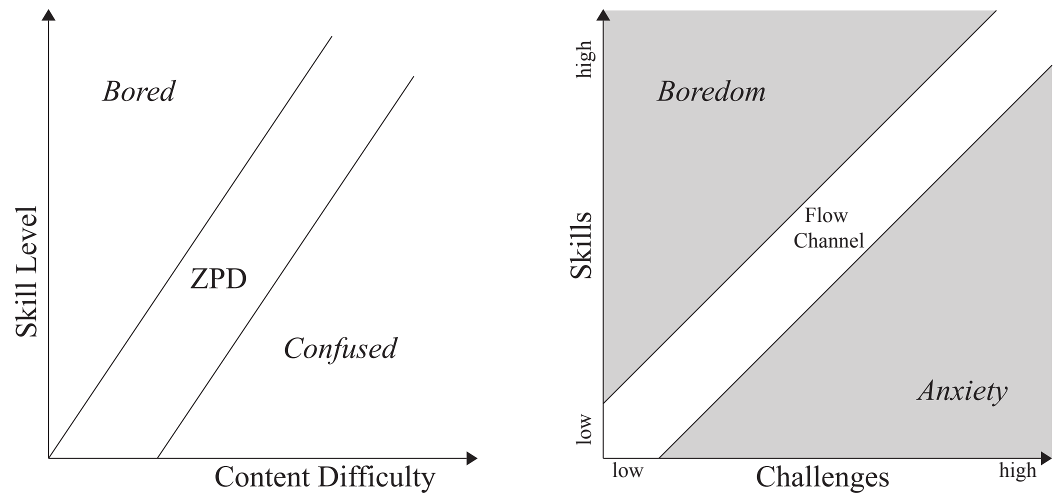

Figure 2.2: (left) the Zone of Proximal Development (ZPD) [46] and (right) Flow [8]

and [31]. Depending on the training domain and the type of knowledge and skills that need to be tracked, some approaches might be more suitable than others. The second challenge is to measure the complexity of the tasks. Several theoretical models for task complexity have been developed ([5], [48]). Dunne et al. (2010) [10] present a concrete but untested computational formula for task complexity within SBT. All these theoretical models determine task complexity based on characteristics of the task: the number of actions that needs to be performed, the relations between the actions, the uncertainty of the outcome of actions, etc. In practice, the values of some of these characteristics might be difficult to determine. Because of this, and, because it might enhance the acceptability of the system by the prospected end users, it might be more appropriate to rely on the experience of domain experts to determine the complexity of a task.

The next two requirements are closely related: scenarios should resemble real-world situations the trainee might possibly encounter and scenarios should be realistic. Peeters justifies both requirements with the claim that they will promote transfer of the learned skills from the virtual environment to the real world. Moreover, relevant scenarios will have meaning to the trainee thereby increasing engagement and his will to learn. In the remainder of this work I will refer to this combined requirement, being relevant and realistic, as the need for scenarios to be authentic to the training domain.

The final requirement is to create varied scenarios. Two arguments are offered for this need. Presenting the trainee with problems of a similar nature but with different details, will help him abstract away from the surface aspects of the problem presented, and to recognise the underlying principles. Secondly, having varied scenarios would increase the replayability of the game, offering the trainee more practice opportunities.

2.2 SBT in Practice

To profit from the experience and knowledge of the prospected end users of the system, I conducted interviews with several instructors who employ SBT in their curriculum. Two First Aid instructors were found willing to participate. They explained that First Aid classes are done in groups, with one instructor teaching several trainees at the same time. Practice is done in the form of role playing with the help of specially trained actors. These actors, called LOTUS-victims, mimic a variety of injuries. LOTUS-victims are a relatively scarce resource and only in exceptional cases will there be more than one available. Furthermore, the instructors often do not know beforehand who the LOTUS-victim will be. Training can take place at a regular training centre or on site at a client.

To obtain a wider perspective I also considered a military domain and interviewed four Helicopter Directive Operator (HDO) instructors of the Dutch Royal Navy (see Figure 2.3). An HDO sits aboard a ship and uses his instruments to instruct helicopter pilots regarding their flight patterns. HDO training is performed using a simulator where instructors can control all relevant aspects of the virtual environment at run-time. Trainees usually have individual training sessions with at least one, but often more instructors to guide them. The training adheres to a strict curriculum with weekly objectives and assessments that the trainee has to pass to be allowed to continue the training.

Furthermore, I have observed and interviewed a military officer who instructs other military officers in designing virtual training scenarios using VBS2 (Virtual Battlespace 2), a virtual training environment designed for military use.

Figure 2.3: Interview with HDO instructors of the OP-school.

Despite the differences between the training domains, the instructors considered many of the same requirements for a good scenario. These requirements are discussed below.

First of all, instructors confirmed that the basis of a scenario design should be determined by the learning objective. If the trainee is known to already be proficient at this objective, additional challenges can be added, but the learning objective should be the focus of the training. First Aid instructors also have to take limitations of the location into account, since they often have little control over their workspace when working on site with the customer. Scenarios have to fit the environment to be realistic. Therefore, if a victim claims to have fallen down a flight of stairs, such stairs need to be in the vicinity. If the training location does not have windows it is impossible to use a window-related incident in the scenario. However, when there is no realistic option available, the learning goal takes precedence over realism and the trainees are asked to imagine the relevant details.

Another requirement put forward by the instructors is that the scenario has to be complete. For HDO training this means that the scenario starts with the take-off after which the trainee has to perform several tasks in flight. At the end of a scenario the helicopter has to return to the ship for landing. In the domain of First Aid, scenarios are not as strictly outlined. However, there are five basic steps that need to be performed in all First Aid situations. One of the instructors emphasised that she always tries to force the trainee to actively consider all five steps. The first step for example is to secure the environment both for the victim and the trainee. To challenge the trainees she would therefore instruct the LOTUS-victim to keep swaying the knife with which he supposedly cut himself while asking for help. It would be the job of the trainee to recognise this as a potential danger and to first ask the victim to put down the knife.

A third aspect to consider during the design of a scenario is the prospective trainee. HDO instructors have a select group of trainees with extensive reports on their individual progress. They can therefore focus the scenarios in such a way as to address any weaknesses of the specific trainee. Although First Aid instructors are not able to focus on one individual, they also take the competency level into account. Moreover, the instructors also consider the background of the group. For example, a group of students might do well when asked questions about the theory, but could require above average amounts of hands-on practice to actually bring that knowledge into practice.

For both domains the difficulty of a scenario is increased by adding additional challenges atop the basic procedures. Sometimes these extra challenges are established by adding obstacles that hamper the trainee in performing his tasks. A First Aid scenario where a victim falls down the stairs can be complicated by giving this victim a backpack, which needs to be removed before continuing with the diagnoses and treatment. In other situations the trainee can be tempted to make a mistake. During the class I observed on creating military training scenarios in VBS2, one of the officers was tasked with designing an ambush situation along a curvy road. For the mission to succeed the trainee had to place himself and his men on the proper side of the road so the oncoming traffic would be less likely to spot the ambush. By placing the trainee at the wrong side of the road at the start of the scenario the exercise could be made more challenging, because now the trainee had to actively pick the right spot. When adapting the scenario to the competency level of the trainee the fit should not always be perfect. The HDO instructors indicate that they sometimes give the trainee a challenge that is purposefully above his expected performance level. The aim is to drag the trainee outside his comfort zone and to introduce a little stress and frustration. On the other hand, if the trainee is too stressed, an easy scenario can be used as a feel-good mission to return the trainee to a positive mind set. This approach is related to the discussion of the ZPD and Flow in the literature section. In this situation, however, the instructors purposefully push the trainees outside the ZPD to teach them to perform their job even if they are stressed and overwhelmed. In First Aid training too, it is customary to occasionally add a little pressure to the training. By masquerading a LOTUS-victim as a trainee who suddenly faints, the real trainees get a feeling of urgency and reality.

Along the same line of thought is the idea that it is important to surprise the trainee once in a while to break the pattern of the curriculum. If, during a First Aid course, all exercises of the day have to do with broken limbs it is useful to insert a totally different injury somewhere during training to ensure the trainees stay focused and do not start making unjustified assumptions.

A final requirement that the First Aid instructors mentioned when asked for their expectations of the system was variety. It is important that a trainee can practise often without being faced with exactly the same scenarios.

2.3 Scenario Requirements

This section concerned an analysis of the domain and the determination of the requirements for scenario generation. The requirements found in the literature were the same as those mentioned by instructors experienced in the use of SBT. Scenarios have to be designed with a specific learning goal in mind, the difficulty level of the scenario should match the competency level of the trainee, and scenarios should be authentic. Moreover, there should also be enough variation between scenarios offered to the same trainee. The instructors also provided additional claims that scenarios should cover a situation from beginning to end and that once in a while the trainee should be presented with a scenario that is too difficult or focusses on unexpected learning objectives. This last requirement will not be considered in the remainder of this work, because the selection of the difficulty level and the learning objective are external to the scenario generation process.

Chapter 3

Background

To address the problem of automatic scenario generation for use in serious games, I propose a framework that combines automated planning techniques with the use of semantically annotated 'smart' objects. This chapter provides the theoretical foundations for this framework and introduces the basic ideas behind these techniques. The first section offers a general introduction to automated planning and the second section explains the concept and value of Smart Objects.

3.1 Automated Planning

The first technique that is used in the scenario generation process is automated planning. This section presents a basic introduction to this subfield of Artificial Intelligence. It draws heavily from the book *Automated Planning, theory and practice* written by Ghallab et al. (2004) [14], an extensive and comprehensive introduction to automated planning, and the work of Nau (2007) [32] which gives an updated overview of the current approaches and trends. According to Ghallab *"planning is the reasoning side of acting"*. The computational study of this deliberation process is called Automated Planning. There are many different kinds of planning problems that can be handled with automated planning. A few examples are path planning, which is concerned with finding a path from a starting position to a goal position; manipulation planning, how to handle physical objects and use them to build assemblies; and communication planning, what and when should be communicated to other agents. Because real world planning problems are very complex, restrictive assumptions are usually made concerning the planning domain. Just as there are many different planning problems, there are also many different planning approaches. To reason about all these different planning problems and techniques Ghallab proposes a conceptual model for planning. This section first explains this conceptual model and then discusses the restrictive assumptions that are made regarding the planning domain. Next, the different planning techniques are reviewed. The technique that is employed by the framework for automated scenario generation is a hybrid approach of two planning techniques, Hierachical Task Network (HTN) planning and plan-space planning. The final part of this section provides a more in-depth explanation concerning Hierarchical Task Network Planning (HTN) and the extension to hybrid planning.

3.1.1 Conceptual Model for Planning

A conceptual model is a tool employed to describe and reason about a problem. Although it can significantly differ from the computational considerations used in solving the problem, such a model can aid in the explanation of basic concepts and assumptions. The Conceptual Model for Planning as presented in [14] is shown in Figure 3.1. As can be seen from the figure the model consists of three primary parts: *a state-transition system* which is a formal model of the real-world system for which the plans should be created; *a planner*, which produces the plans; and *a controller*, which performs the actions according to the plan, in order to change the state of the system. The following paragraphs discuss the seperate components in more detail.

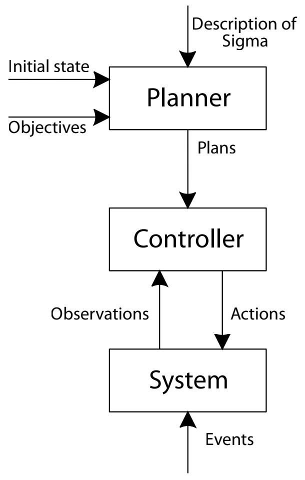

Figure 3.1: The Conceptual Model for Planning as presented by Ghallab (2004) [14].

A state-transition system is formally defined as a 4-tuple $\Sigma = (S, A, E, \gamma)$, where

$S = s_0, s_1, s_2, ...$ is a set of states;

$A = a_1, a_2, \ldots$ is a set of actions;

$E = e_1, e_2, \ldots$ is a set of events;

$\gamma : S \times (A \cup E) \rightarrow 2^S$ is a state-transition function.

As can be seen from the state-transition function, both actions and events can change the state of the system. The difference between an action and an event lies in whether the planner has any control over them or not. Actions are performed by the controller. Events, on the other hand, are contingent. They represent the internal dynamics of the system and cannot be triggered or chosen by the controller. An action a is said to be *applicable* to state s if $\gamma(s, a)$ is not empty. If e is an event and $\gamma(s, e)$ is not empty, e may *possibly* occur when the system is in state s.

State-transition systems are often represented by a directed graph whose nodes represent the states in S. If $s' \in \gamma(s, e)$ where $e \in A \cup E$ is an action or event, then there is an edge between the nodes s and s', which is labeled with the action or event. Given a state-transition system Σ and an initial state, the purpose of planning is to find which actions to apply in which states to achieve some objective. Objectives can take several forms. The simplest form is that of a specific goal state s_g or set of goal states S_g that needs to be reached. Other objectives might have more complex requirements including states that need to be reached, states that need to be avoided, and optimisation rules.

The planner takes as input a *planning problem*: a description of the state-transition system Σ, an initial state, and an objective. It outputs a plan or policy that solves this planning problem. A plan is a sequence of actions, whereas a policy is a partial function from states into actions. A distinction is made between *offline* and *online* planners. An offline planner does not receive feedback about Σ's current state. It generates a plan which is then executed by the controller without any intervention from the planner. In case of unforeseen situations the plan cannot be updated. In most real-world applications there are differences between the physical system and the model (Σ). If the controller is unable to handle these differences, a link from the controller to the planner is required. The planner then becomes an *online* planner, capable of dynamic planning. An online planner is informed of the execution status, and planning can be interleaved with acting by introducing mechanisms for plan supervision, plan revision, and replanning.

The controller takes as input the plan and observations about the current state of the system. These observations can be modeled as an observation function $\eta : S \rightarrow O$ that maps states to observations. If this function is a one-to-one mapping, the controller can use the observations to deduce the exact state of the system at the time of observation. In this case the observations provide *complete* information. If η is not a one-to-one mapping of states into observations, the controller can only identify a set of possible states the system might be in. This is called *incomplete* information. Based on the plan and the observations the controller outputs actions to be performed in the state-transition system.

3.1.2 Classical Planning

Automated planning for real world applications can become exceedingly complex. Therefore, it has long been customary to make restrictive assumptions regarding the planning domain. This resulted in the definition of classical planning problems. Classical planning is concerned with planning problems for domains that can be modeled by a restricted state-transition system. Such a system is a state-transition system as described above, with the addition of the following restrictive assumptions:

Assumption A0 (Finite Σ) The system Σ has a finite set of states.

Assumption A1 (Fully Observable) The observations the controller receives concerning the current state of Σ provide complete information.

Assumption A2 (Deterministic Σ) The execution of a specific action in a particular state will always lead to the same result state.

Assumption A3 (Static Σ) The set of events E is empty, meaning that the system stays in the same state until the controller initiates a state transition by performing an action.

Assumption A4 (Restricted Goals) The planner objectives can only take the form of a specific goal state or a set of goal states.

Assumption A5 (Sequential Plans) The solution to a planning problem is a plan in the form of a linear sequence of actions $a_1, a_2, \ldots$. Consequently, no partially ordered plans are allowed.

Assumption A6 (Implicit Time) Actions have no duration meaning that state transitions happen instantaneously.

Assumption A7 (Offline Planning) The planner works offline. The plan is generated based on the planning problem, and executed without any intervention by the planner.

This restricted model is denoted $\Sigma = (S, A, \gamma)$ instead of $\Sigma = (S, A, E, \gamma)$, since there are no contingent events. Although the assumptions of the restricted model are very strong and not applicable to real world problems, the study of classical planning has aided in the formalisation of the field and its methodologies, and in the development of algorithms and techniques that can be scaled to more realistic problems.

Planning in such a restricted model might appear trivial since the problem can be reduced to finding a path in a graph, a well-known and well-solved problem. However, the challenge lies in the definition and representation of this graph. Even for simple domains an explicit representation of Σ can become many orders of magnitude greater than the number of particles in the universe ([14]). Powerful implicit representations are therefore required that can describe the search space in a compact and easy to search format. Several representations have been designed, but I will only consider the classical representation. In this representation a state of the state-transition system is a set of ground atoms in a first-order language that contains no function symbols. If g is a set of literals, then state s is said to *satisfy* g if there exists a substitution σ such that every positive literal in $\sigma(g)$ is in s and no negated literal of $\sigma(g)$ is in s. Actions are instantiated planning operators. A *planning operator* is defined as a triple $o = (\text{name}(o), \text{precond}(o), \text{effect}(o))$ where:

name(o) uniquely identifies the operator and names the variables used anywhere in o.

precond(o) is a set of literals that have to be present in a state before o can be applied.

effect(o) is a sets of literals that are added or removed from the state after the execution of o.

For non-classical domains there exist extensions to this representation to deal with the relaxation of the assumptions discussed earlier.

3.1.3 Types of planners

Planning algorithms can be classified into the following three categories based on if and how they use domain-specific knowledge: domain-specific planners, domain-independent planners, and domain-configurable planners. *Domain-specific* planners are tailor-made for use in a given planning domain. They make extensive use of domain-specific knowledge to guide and speed up the planning process. These planners often have the highest performance but cannot be applied to other domains without major modifications. *Domain-independent* planners, on the other hand, do not use any domain knowledge. This allows them to cover a large set of domains without any modifications. However, these types of planning algorithms are often quite inefficient since they have limited opportunities to prune the search-space. As a result, these planners are only feasible in simplified domains. In practice, domain-independent planners are therefore only applied to classical planning problems or other restricted models. *Domain-configurable* planners try to find a balance between the coverage of domain-independent planners and the performance of domain-specific planners. They employ a domain-independent planning engine but use domain-specific knowledge to constrain the search-space considered by the engine.

In the following sections an overview is presented of the best-known approaches to automated planning. The first four sections discuss approaches that are domain-independent, although many modified implementations exist that make use of domain-specific heuristics. Domain-configurable planners can be divided into two types: Hierarchical Task Network planners and Control Rule planners. These two categories are considered in the last two sections. No domain-specific planners are discussed since they depend on specific details of the problem domain for which they were constructed.

State-Space Search

The basic idea behind state-space search algorithms is straightforward. These algorithms search for a solution to the planning problem directly in the state space Σ. Each node in the search graph represents a state, each arc a state transition. Based on the implementation of the search process, state-space algorithms can be divided into two categories: forward search and backward search. Forward search planners start from the initial state and non-deterministically choose and apply an action to the current state until either the goal state has been reached, or there are no more actions applicable. Backward search starts from the goal and non-deterministically applies the inverse of the planning operators until the initial state has been reached. Both methods are sound and complete. There are pruning techniques and heuristics available to improve the efficiency of search-space planners, but for a long time these heuristics were either not efficient enough or made the algorithms incomplete. Only when it was realised that heuristics could be computed using other approaches such as the Planning Graphs discussed below, state-space techniques became popular.

Plan-Space Search

As the name suggests, plan-space planners do not look for a solution in the state-space Σ but in the plan-space. In this space every node represents a partially specified plan, and every arc is a plan-refinement operation. A partially specified plan is not a simple action sequence. Instead, it consists of a set of partially instantiated actions together with constraints on the ordering, variable instantiation, and causal links between these actions. A causal link between two actions states that the first action supports the preconditions of the second action. These links and constraints are required for plan-space planning, because states are not explicit and the planner has no notion of the current state. Without additional information the planner would not be able to determine whether or not an action in the plan still has open preconditions. The basic idea behind plan-space planners is to start with an initial plan that only contains dummy actions encoding the propositions of the initial and goal state. Unless the planning problem is trivial this initial plan will contain flaws. Flaws can be actions whose preconditions have not been established, or threats to causal link constraints. A plan-space algorithm iteratively selects a flaw to address and then non-deterministically chooses a resolver to solve this flaw. A resolver is either the addition of a causal link between actions already in the plan, or the addition of an action that supports the open preconditions. The order in which flaws are addressed is very important for the efficiency of the algorithm, and varies between implementations. One of the basic principles of plan-space planners is the idea of *least commitment*. Constraints are only added if strictly necessary. This will result in more general solutions that might not specify the exact ordering between, or variable bindings of, the actions. For some time, plan-space planners performed significantly better than state-space planners. However, a huge drawback of the plan-space approach is that the planner does not model states and therefore has no notion of the current state. Recent state-space planners have been able to profit from domain-specific heuristics requiring information on the current state. These heuristics have enabled state-space planners to scale up to large problems that are not feasible for plan-space planners. Plan-space planners are still interesting because they do allow the construction of partially ordered and partially instantiated plans, and have explicit causal links, making the plan easy to understand for a user.

Planning Graphs

Planning graphs are a tool to significantly reduce the search space of a planning algorithm. A planning graph consists of several levels. Level n contains all actions a whose preconditions are satisfied by level $n-1$ and do not violate mutual-exclusion constraints. The literals of level n are all the literals of level $n-1$ plus all effects of all actions at level n. This set of literals does not need to be consistent, and a level can contain conflicting actions. A basic graph planner iteratively builds a larger planning graph whilst performing a state-space search that includes only those actions that are in the planning graph, until a solution is found. A planning graph can be constructed in polynomial time, and by restricting the search space of the state-space algorithm to the planning graph, the algorithm's performance is dramatically improved.

Translation Into Other Problems

The basic idea of this approach is to use problem solving techniques from other fields to solve planning problems. First the planning problem is translated into another kind of combinatorial problem such as satisfiability or integer programming. This problem is then solved using the appropriate methods and the resulting solution is translated back into a plan. For specific examples and more details on the translation step the interested reader is referred to [14].

Hierachical Task Network Planners

Hierarchical Task Network planners differ from the other planning techniques discussed so far, in that the planner's objective is not a set of goal states, but rather a collection of tasks that need to be performed. Using domain specific rules, called methods, these tasks can be decomposed into subtasks, which can be decomposed further until primitive tasks are reached. These primitive tasks form the solution to the planning problem. Section 3.1.4 discusses HTN planning in more detail.

Control Rule Planners

Control rule planners focus on pruning the search space. They contain a set of domain-specific rules that describe conditions under which the current node can be pruned from the search space. The control rules are often specified using a simplified form of temporal logic, describing relations between the current state and subsequent states. Most implementations rely on forward search state-space planners. If the planner reaches a state that can be pruned according to the control rules, it backtracks and tries a different option. Similar to the use of a planning graph, the control rules drastically reduce the search-space and improve the efficiency of the search algorithm.

3.1.4 HTN planning

Hierarchical Task Network (HTN) planning has been more widely used for practical applications than any of the other planning techniques described. This is partly because the HTN format provides a convenient and intuitive way to represent expert knowledge, and uses this knowledge to reduce the size of the search space. A few examples of HTN planner implementations are SHOP2 [33], O-Plan [9] and UMCP [11]. As explained above, HTN is an approach to planning that does not focus on reaching specific goal states, but rather on performing specific tasks. Two types of tasks are discerned: *primitive tasks* and *non-primitive tasks.* Primitive tasks are ungrounded actions and can directly be executed by the controller. Non-primitive tasks need to be decomposed into primitive tasks using domain specific rules called *methods.* An HTN method is defined as a 4-tuple: $m = (name(m), task(m), subtasks(m), constr(m))$ where:

> name(m) is an expression of the form $n(x_1, \ldots, x_k)$ where n uniquely identifies the method and $x_1, \ldots, x_k$ are all of the variable symbols that occur in m.
>
> task(m) is the non-primitive task that the method can decompose.
>
> subtasks(m) is a set of tasks into which task(m) is decomposed using this method.
>
> constr(m) is a set of constraints on the subtasks.

Together, the subtasks and constraints form a *task network.* A task network is defined as a pair $w = (U, C)$ in which U is the node set and C is a set of constraints. Each node $u \in U$ contains a task t_u. Different HTN implementations are able to handle different kinds of constraints. The most commonly used constraint is a precedence constraint. Precedence constraints are written as $u \prec v$, where u and v are tasks, meaning that u and every descendant of u should be performed before v or any descendant of v. Other common constraints are the before, after, and between-constraints that specify a literal that has to be true in, respectively, the state that occurs just before a task or its descendants, in all the states between two tasks, or in the state occurring just after a task.

An HTN planning problem is defined as a 4-tuple $\mathcal{P} = (s_0, w, O, M)$, where s_0 is the initial state, w is the initial task network, O is a set of operators and M is a set of HTN methods. A plan π now consists of a sequence of actions obtained by decomposing w into π in such a way that the plan is executable in state s_0 and adheres to all constraints specified in w. The basic idea behind HTN planning is thus to recursively replace non-primitive tasks with appropriate task networks, and to instantiate primitive tasks with actions until there are only actions remaining. Additional bookkeeping mechanisms are required to ensure that the decomposition process adheres to the constraints. An important consideration for such mechanisms is whether the domain is *totally ordered* or *partially ordered.* In a totally ordered domain the HTN methods specify a total order on their subtasks, i.e. precedence relations are specified between all possible tasks. In a partially ordered domain, some precedence constraints can be specified, but they are not required. Requiring a total order on the task network might seem overly restrictive, but it does allow for efficient forward search planner implementations such as [33]. These implementations plan actions in the order in which they are executed. This enables the planner to know the current state of the system during the planning process allowing for easy checks on the constraints. Partially ordered domains require more complex bookkeeping mechanisms but do offer the opportunity to interleave actions of different tasks. Figure 3.2 shows the difference between the two approaches. The top schema shows a totally ordered domain where actions cannot be interleaved because of the precedence constraints on their parents. In the bottom schema, the precedence constraints between the subtasks 'get(p)' and 'get(q)' are removed, allowing their subtasks to be interleaved.

3.1.5 Hybrid planning

HTN planners are popular because they use the knowledge of domain experts to guide the search. However, most real-world applications tend to be merely "partially hierarchical", i.e. available decomposition schemas cover only part of the domain. To support planning in such situations hybrid planners have been designed.

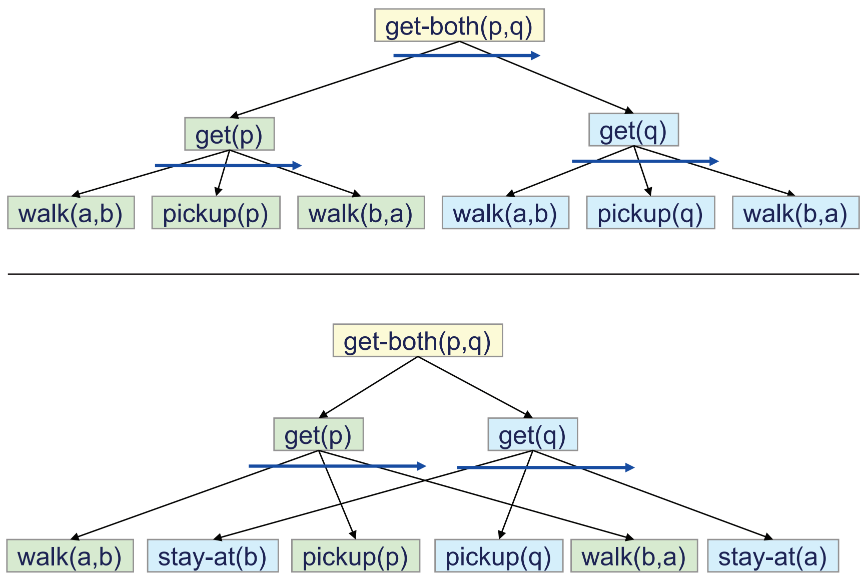

Figure 3.2: Schematic overview of the decomposition process of a high-level task into actions. The top schema shows a totally ordered domain with precedence constraints (represented as blue arrows) specified for all subtasks. The bottom schema shows how the decomposition process can interleave actions from different subtasks, if the domain is partially ordered. These images are copied from the slides accompanying the book of Ghallab et al. [14].

These planners apply HTN methods to reduce non-primitive tasks where possible, and resort to primitive action planning methods (i.e. methods that only reason with primitive tasks, such as state-space or plan-space planning) where necessary. A few examples of hybrid planners are: *HyHTN* [30], the *Duet* planner [13] which combines the HTN planner SHOP2 with the graph planner LPG, and *DOPCL* [49] which extends the plan-space planner SNLP to work with non-primitive tasks.

Altough they do not offer a concrete implementation, Kambhampati et al. (1998) [22] describe an elegant framework for hybrid planners. The authors discuss the combination of HTN and plan-space planning techniques, and identify the technical challenges that occur regarding systematicity, preservation of user-intent, and parsimony of solutions. A planner is called systematic if it does not consider the same plan or partial plan more than once. This is a desirable property for efficient planners. In hybrid planners this property can be easily lost. For example, if the planner could choose between two tasks, t_1 and t_2, to establish an open precondition, but t_1 is actually a subtask of t_2, then considering both tasks as candidates would result in overlapping plans. The reduction of t_2 into t_1 would mean that the planner considered this partial plan twice and would therefore no longer be systematic. To avoid this problem the planner should only consider tasks to establish a certain precondition if they are not a subtask of any other task that could achieve the same effect. Tasks that satisfy this requirement are called *maximally abstracted*. The second issue, the preservation of user-intent, is similar in nature. If the planner is allowed to consider tasks that are subtasks of other tasks, the planner might skip a decomposition step. For example, consider a travel planner and let t_1 be the task of actually getting in the train, and t_2 the high-level task of travelling by train. The task t_2 would be reduced to buying a ticket, getting in the train (t_1), and getting out the train. If the planner is allowed to consider t_1, it might skip the task of buying a train ticket, thereby violating the user's intent. The final challenge is to keep the plans simple, without redundant actions. This problem is actually induced by forcing the planner to only consider maximally abstracted actions. Although this rule ensures systematicity and the preservation of user-intent, it also means that the planner will select high-level tasks to establish secondary effects. If for example, the goal is to simply have a ticket (but not to travel), the planner will still select the maximally abstract task and include travelling actions in its plan. This problem can be addressed by specifying primary effects of a task. A task might only be considered to establish a precondition if this condition is contained within its primary effects. A final consideration that should be taken into account in hybrid planning is phantom establishment. This is the idea that since high-level tasks do not advertise all effects of their recursive subtasks, one of these subtasks might still establish an open precondition. The planner therefore will have to make allowance for preconditions to remain open until all non-primitive tasks have been decomposed into primitive tasks.

In conclusion it can be said that hybrid planners make effective use of the domain knowledge available, and are still powerful enough to handle difficult real world problems. With the use of plan-space or other action-planning techniques, hybrid planners are capable of working in partially hierarchical domains where HTN planners cannot function optimally.

3.2 Smart Objects

This section discusses the concept of Smart Objects, the second ingredient of our solution to automated scenario generation. Smart Objects are virtual objects that are annotated with semantic data to give meaning to the object. Before going into the details of these annotations it is important to consider the use of semantic knowledge in a virtual environment. Most virtual environments scarcely model any semantic information and consist of little more than annotated geometric representations of the environment and of the objects embedded in it ([44]). This lack of knowledge inhibits the automatic generation of environments, meaning that creating a virtual world is a manual, time and labor-intensive job. Moreover, the moment the designer places an object in the world, its meaning and thereby the designer's intent are lost, making it difficult to iteratively build an environment or re-use specific parts. Adding semantic information to the game world can ease this process. Tutenel et al. (2008) discuss the role of semantics in virtual worlds [44]. They discuss examples that show how semantic annotations can offer advantages both in the design phase and during runtime. At runtime semantic knowledge can facilitate realistic interaction between objects and agents without having to update the agent's representation when a new object is added. Moreover, semantic annotations can support adaptivity in games. The environment can, for example, adapt to the weather conditions or react differently based on the actions of the trainee. In the design phase the addition of semantic data can ease manual creation of virtual worlds. Given the appropriate semantic knowledge, the design environment could for example automatically place streetlights along the roads laid out by the designer. Finally, and most importantly given the purpose of our research, Tutenel also shows how semantic information could be used to facilitate procedural content modelling techniques in the automatic generation of game environments. Examples include the generation of virtual terrains by using detailed soil quality, and the generation of house interiors that attach functions to each room in the house, such as living room, kitchen and bedroom, to determine its relative size. It is clear from these examples that enhancing the virtual world with semantic knowledge can definitely have its advantages.

Given the new and interesting possibilities semantic annotations offer, the question becomes how to add this information to the world. A succesful approach to annotating the virtual world with semantic knowledge can be found in Smart Objects. The term Smart Objects was introduced by Kallmann and Thalmann (1998) in their work on guiding the interaction between objects and virtual human agents in a virtual world [21]. They call an object 'smart' if it has the ability to describe its possible interactions. They point out that even simple object-agent interaction tasks, such as opening a door, can become rather complex if the agent has to do all the reasoning himself. Therefore the authors propose to include, within an object description, additional information regarding its interaction possibilities. In their framework objects contain:

- Intrinsic object properties: detailing the properties that are part of the design of the object such as weight and centre of mass, but also a text description of the general object purpose. For example, an automated door has a general 'door' purpose.

- Interaction information: informs an agent on how to interact with the object, and provides the appropriate parameters for animation, such as positions and hand gestures. Continuing the door example, this information would describe that the door offers the agent the ability to pass through it.

- Object behaviours: describing the possible behaviours of the object and the conditions they are subject to. For the automated door this would be that the door opens if someone is in front of the door, and that the door closes only if the door is currently open and nobody is passing through.

- Expected agent behaviours: these are associated with the object behaviours and describe how an agent should behave during the interaction. In the example this information would indicate where the agent has to be to open the door, and give him footsteps that he can follow to go through the door.

Complex objects can be modelled using several simpler objects that each offer their own interaction possibilities. When a specific task needs to be executed in the virtual world, a reasoning module searches the Smart Objects for a behaviour description matching the task. Once a behaviour has been selected, the reasoning module directs the behaviour of the agents accordingly. In the use of Smart Objects the advantages associated with the enrichment of the virtual world with semantic knowledge become apparent. During runtime these Smart Objects decentralise animation control and seperate high-level planning from low level object reasoning. They also facilitate reusability in the design phase.

The implementation of Smart Objects as discussed above, however, still focuses mostly on facilitating realistic animations. Several other researchers have extended the concept of Smart Objects to reach beyond this animation intelligence. Two extensions that are relevant to this work are those of Abaci and Thalmann (2005) [1] and of Lopes and Bidarra (2011) [26]. Abaci and Thalmann suggest extending the concept of Smart Objects with planning and AI related data. This allows the agents to perform complex interactions with objects, without requiring any specific object knowledge. Instead, an agent can ask the Smart Object for any information he needs. For this purpose, every interaction possibility is extended with preconditions that have to hold before the action can be executed, and the effects on the world after the interaction has finished. The authors illustrate their approach using an example where two agents have to move two boxes to another room. One of the boxes is heavy and requires two agents to be moved. This information is added in the preconditions of the move interaction possibility and forces an agent to recruit the aid of another agent before he can move the box.

The second interesting extension is suggested by Lopes and Bidarra (2011) [26]. The authors aim to use Smart Objects to generate personalised content for games, both serious and non-serious. With each object they associate the experience it can offer a player of a certain type. In a First Person Shooter, for example, some players can be described as ‘explorers’. Certain items or characteristics of the world will lead to a more boring experience for this type of players, whereas other objects might increase excitement. For example, a time bomb leaves less time to explore, discouraging the player from what he really likes, whereas hidden chambers make the world more interesting and exciting. In serious games the same idea can be applied. For example, in a driving simulator the introduction of a steep hill might offer a trainee a more challenging experience. Lopes and Bidarra propose a game content generator that uses these object annotations to select appropriate content for the player based on a player model that is updated during the game.

These two extensions show that the concept of Smart Objects can be used to include any information necessary for a specific application. However, care must be taken that the annotation process will not become too labor-intensive. To promote the reuse of Smart Objects, a general format should be specified where different applications can load different modules that contain data relevant to the application. A first step towards such a framework is made in [20]. Over time, extensive libraries could be created with Smart Objects that can be reused in various applications. This will greatly ease the creation of virtual environments, and allow for realistic interactions between agents and objects without the need to update the agent descriptions.

Chapter 4

Scenario Generation Framework

This chapter presents the framework for a system that automatically generates personalised training scenarios for use in serious games. A concrete implementation of the Scenario Generator is discussed in the next chapter. The automated scenario generation process consists of two main steps. First, given specific learning objectives, the system selects tasks the trainee has to perform. Secondly, based on the tasks, it determines appropriate content for the game world. The task selection is performed using automated planning techniques (see Section 3.1) while Smart Objects (see Section 3.2) facilitate the content selection. Together, these techniques allow the creation of scenarios that *a*) focus on the learning objective; *b*) are adapted to the competency level of the trainee; *c*) constitute a complete, coherent and authentic set of events; and *d*) offer the trainee a wide variety of experiences to practise with. The following section first discusses the context in which the system is expected to function and the resulting constraints and additional requirements. Then, an overview of the system is presented, followed by detailed descriptions of the separate parts. In conclusion, the adherence of the system to the scenario requirements presented in Section 2.3 is evaluated.

4.1 Scenario Generation for Adaptive Educational Games

The Scenario Generator is designed to work within the context of an agent-based Adaptive Educational Game (AEG) ([37], [38]). An AEG guides the trainee during autonomous training sessions. It monitors the trainee's performance and adapts the game to match this performance. This adaptation can take place during the course of the scenario, for example by changing the difficulty or providing hints; or in between the scenarios by selecting an appropriate learning objective for the trainee. The inclusion of the Scenario Generator allows an AEG to even better adapt the training sessions to the needs of a individual trainee, since it can generate personalised scenarios instead of having to find the best match in a static library.

The incorporation of an Scenario Generator within an AEG places constraints and requirements on both systems. First of all, I only consider agent-based AEGs. Such games employ intelligent virtual agents to play the role of human and non-human entities during the scenario. These agents are capable of autonomous reasoning, meaning that they can formulate their own plans to achieve specific goals and react naturally to (unexpected) actions of the trainee. Although it is possible to guide the behaviour of the agents, it is undesirable to give them too specific instructions, since these might interfere with the believability of the agents' characters. Therefore, agent behaviour is preferably specified using high-level goals. Another point to consider is the diagnostic property of SBT (see Section 2.1). One of the advantages of SBT is that a predefined scenario makes it easier to assess the trainee, because observers know what is going to happen and when. In order to preserve this diagnostic property, the automatically generated scenarios will have to provide the AEG with similair information regarding the expected behaviour of the trainee. On the other hand, the Scenario Generator places certain requirements on the AEG. First of all, the Scenario Generator requires information on the tasks the trainee needs to practise. It is important to notice that a task is not equal to a learning objective. A learning objective might be formulated as the trainee's capability of successfully performing a specific task, but it can also refer to a more general skill. Moreover, to match the scenario to the performance levels of the trainee, the Scenario Generator must be able to request these performance levels. The AEG will therefore require some kind of Student Model that tracks the performance of the trainee. Furthermore, to reason about the tasks the trainee is to perform, and to select the appropriate content, the Scenario Generator must have access to the domain knowledge encoding the tasks and available objects. It is therefore assumed that the AEG system contains some form of Wold Knowledge Base that contains this data. Finally, the Scenario Generator assumes that the AEG is capable of generating a virtual environment based on the minimal content requirements the Scenario Generator outputs. This is one of the reasons why the Scenario Generator considers Smart Objects, since they can facilitate procedural content generation [42].

To recapitulate, the AEG system requires the Scenario Generator to output high-level goals for the virtual agents and information on the planned actions of the trainee. The Scenario Generator expects the AEG to provide the task to train and information regarding the trainee's skills and relevant domain knowledge.

4.2 Overview of the Scenario Generator System

This section provides an overview of the system. It details the requirements, components, and the in- and output of the system.

Based on literature studies and interviews with experts, the following list of requirements has been compiled for effective training scenarios (see also Section 2.3):

- The scenario needs to be *focused on the learning objective.*
- The scenarios must be *adapted to the competency level of the trainee.*
- The scenario must constitute a *complete exercise.*
- The scenario has to be *authentic.*
- Scenarios have to be *varied.*

Moreover, to work within the context of an AEG system the Scenario Generator should output: the actions the trainee is expected to perform; a list of the minimum required game content; and high-level goals to guide the behaviour of the virtual agents during the scenario.

To generate scenarios that fulfil all these requirements, a system design as shown in Figure 4.1 is proposed. The system consists of two main components, an Action Planner and an Object Selector. The Action Planner is responsible for generating a complete and coherent action plan for the trainee that offers him the possibility to practise the target task at an appropriate difficulty level. To determine an action plan for the trainee the Action Planner employs a hybrid HTN technique. It takes a scenario template, consisting of a sequence of high-level tasks, as input and guides the decomposition process to ensure that the target task is included, and that the trainee is challenged but not overwhelmed. The plan that is generated for the trainee imposes certain requirements on the game world, both on the objects that have to be present and on the behaviour of the agents. It is the job of the Object Selector to ensure that these requirements are met. The employment of Smart Objects enables the Object Selector to reason about the interaction possibilities an object can offer the trainee, if it were to be placed in the game world. The interaction possibilities are called *services.* Different Smart Objects can offer the same service. For example, a lighter and a match can both offer the service 'fire'. Agents are considered to be special kinds of Smart Objects that can offer more complicated services. Some services have specific requirements of their own. Sometimes these requirements take the form of other services that have to be present and in other cases they may specify conditions that have to be achieved by an agent. In the latter case the Object Selector turns for help to the Action Planner to establish these preconditions and determine goals for the agents.

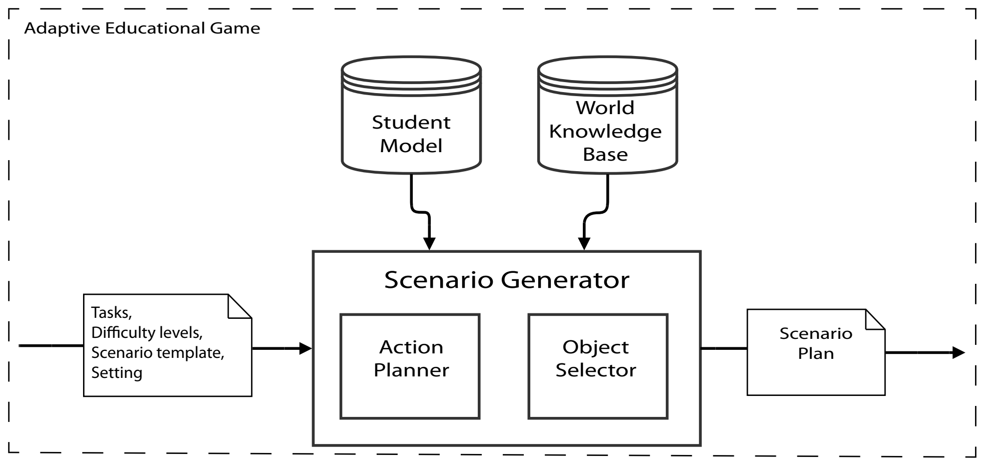

Figure 4.1: An overview of the Scenario Generator system.

Input

As can be seen in Figure 4.1 the Scenario Generator requires the following input:

- *Task(s)*: the target tasks that the trainee must practise during the scenario.
- *Difficulty level*: the desired difficulty levels for each target task. This level is defined as the skill level that is assumed necessary to successfully perform the associated target task. A skill level is represented as a number between 0 and 1, where 0 indicates no skill and 1 represents mastery.
- *Scenario Template*: an ordered list of high-level tasks that together constitute a complete training exercise. The Scenario Generator uses this template to ensure the "complete exercise" requirement.
- *Setting*: an identifier for the desired setting or location of the scenario. The setting is used to ensure authenticity and influences the object selection process.

During the scenario generation process the Scenario Generator also requires access to the skill levels of the trainee, and domain knowledge regarding the tasks and the available world content.

Output

The output of the Scenario Generator is a scenario plan that consists of three components:

- *Trainee Action Plan*: a partially ordered list of actions that the trainee is expected to perform.
- *Agent Goals*: a list of high-level goals for the virtual agents.
- *Smart Objects*: a list of the Smart Objects that have to be present in the game world. The objects are annotated with parameters that influence the difficulty level at which services are offered by the instances placed in the virtual environment.

4.3 Action Planner

The goal of the Action Planner is twofold. First of all it has to determine an action sequence for the trainee which will allow him to practise the target task, and which constitutes a coherent and complete training exercise. Secondly, the planner has to fulfil the open preconditions that the Object Selector poses. To accomplish this task, the Action Planner employs hybrid HTN plan-space planning techniques. When planning the actions of the trainee, the task reduction mechanism inherent to HTN planners constrains the search space and ensures that the resulting solution follows the correct procedures. This is especially important in training situations where the trainee should not merely perform the task, but should perform it correctly executing all the required steps. The plan-space techniques are used to fulfil the preconditions of HTN methods and of the objects selected by the Object Selector.

Before the actual planning process can be detailed, it is important to consider the planning domain. Automated planning for real world applications can easily become very complex. Therefore, it is necessary to carefully consider the assumptions that can be made regarding the domain. This will be adressed in the next section.

4.3.1 Planning Domain

Section 3.1.2 discussed the restrictive assumptions that are made in Classical Planning problems ([14]). In this section we will analyse the current planning domain with respect to those assumptions, and consider the relaxations we have to make and how they can be handled.

Assumption A0(Finite Σ). Whether or not the set of states is finite will depend on the actual training domain. Some domains might require continuous variables to represent state characteristics. Such situations can often be handled by discretising the continuous variables into intervals. Furthermore, HTN planning is able to deal with large state sets because of the search space reductions imposed by the task reduction methods.

Assumption A1 (Fully Observable). Since the current system merely uses the generated plan as a basis for the scenario design there is no real controller as defined in Section 3.1.1 to observe the world. This also means that we do not have to consider how to handle partially observable domains.

Assumption A2 (Deterministic Σ). In the current design I have assumed a deterministic world. In training domains where this assumption is unrealistic, the system can be extended to handle uncertain outcomes. Only uncertainty regarding the successful interaction between the trainee and the Smart Objects is relevant for the Scenario Generator. Such uncertainty might occur when the use of a service by the trainee may fail independent of the correctness of the actions of the trainee. If, however, the Smart Object representation is extended in such a way as to allow the planner control over the outcome of all non-deterministic services, the uncertainty is removed. The system will decide whether or not an action will succeed, and selects and instructs the objects accordingly.

Assumption A3 (Static Σ). This assumption holds, although it might be slightly confusing to see. The planner actually plans the actions of the trainee. From the trainees point of view the actions of the other agents constitute the events. However, the planner controls the goals of these agents and therefore does not consider these actions to be events.

Assumption A4 (Restricted Goals). The goals we set for the action planner do not have the form of a specific goal state (or a set of goal states), but are represented as high level tasks that have to be decomposed. Since this format is inherent to HTN planners this relaxation is automatically accomplished.

Assumption A5 (Sequential Plans). Given the current definition of a scenario plan this assumption is invalid; plans can be partially ordered. Although some of the more efficient HTN implementations do not support partially ordered plans, the basic HTN approach does. Given the inherent partial order nature of the plan-space planning approach, hybrid plan-space HTN planners in particular should be able to deal with partially ordered plans.

Assumption A6 (Implicit Time). Whether this assumption holds depends on the specific training domain. However, we will assume that for the purpose of determining the game world content, the duration of actions does not need to be considered. If this is not the case there are several HTN implementations that are able to handle durative actions [7]. Most of these approaches maintain annotated state representations and consider additional time constraint-techniques that could also be applied to the current system.

Assumption A7 (Offline Planning). This assumption is valid, the planner generates the scenario plan beforehand and does not adapt the plan during the execution of the scenario.

From the above analysis it can be concluded that HTN planning techniques will be adequate to deal with the relaxations that have to be made regarding the assumptions associated with Classical Planning.

4.3.2 Planning Process

The planning process forms the basis of the Scenario Generator. Figure 4.2 shows a high-level flowchart of the process. Before explaining the details of the different steps, the definition of an HTN method as given

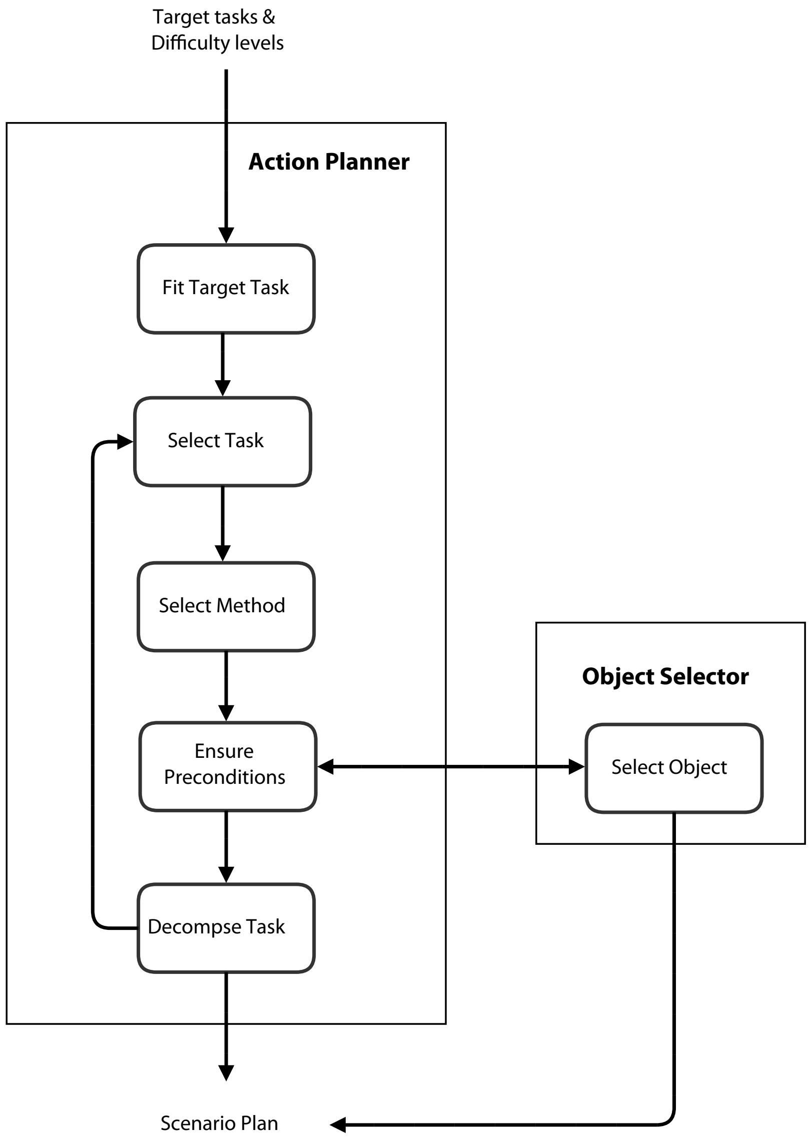

Figure 4.2: Flowchart depicting the planning process of the Scenario Generator.

in Section 3.1.4 has to be extended.

In Section 3.1.4 an HTN method was defined as a 4-tuple: $m = (name(m), task(m), subtasks(m), constr(m))$ where:

> name(m) is an expression of the form $n(x_1, \ldots, x_k)$ where n uniquely identifies the method and $x_1, \ldots, x_k$ are all of the variable symbols that occur in m.
>
> task(m) is the non-primitive task that the method can decompose.
>
> subtasks(m) is a set of tasks into which task(m) is decomposed using this method.
>
> constr(m) is a set of constraints on the subtasks.

In the presented framework this definition is extended with two components: a vector of difficulty levels and weights, and a set of preconditions. The difficulty levels are defined equal to those given as input to the Scenario Generator. Each difficulty level is associated with a specific subtasks specified in the method. The difficulty level for a subtask is a number between 0 and 1, indicating the skill level the trainee has to possess in order to successfully complete the subtask. The associated weights indicate the influence that the difficulty level of a subtask has on the difficulty of the parent task. Therefore, the weights of all subtasks have to sum to one. The difficulty levels and associated weights are determined by human experts, since this seems a more reliable approach than the computational methods for task complexity as briefly discussed in Section 2.1.

The second additional component is a set of preconditions. These conditions specify services that have to be offered by the game world before this method can be applied. For example, the method that decomposes the task "ensure safety" into "extinguish fire" is only applicable if there is a fire in the game world.

Fitting of the target task

The most important consideration for a training scenario is that it offers the trainee the opportunity to practise the target tasks. These tasks can be high-level tasks that occur directly in the scenario template, or they can be lower level tasks that can only be introduced into the plan by decomposing the template tasks. In the first case ensuring the occurrence of the target task in the scenario is trivial. In the second case however, the decomposition of the template tasks will have to be guided, i.e. the target tasks will have to be 'fitted' within the template. To fit these target tasks the system has to know which template tasks can be decomposed into the target tasks, and what the exact sequence of HTN methods is that has to be applied. Since this information is static, it can be computed once for every possible combination of scenario template and target task, and used in all subsequent scenario generation processes. If there are no decomposition sequences that allow the target task to appear in the scenario the Scenario Generator returns failure. If there is more than one decomposition sequence available, the Action Planner chooses randomly between all sequences that best match the desired difficulty levels for the scenario. The "method selection" step details the computation of the match between the difficulty of a method and the desired difficulty levels. For a sequence of methods this measure is taken to be the mean of the values of the individual matches.

Task Selection

Once an HTN method sequence has been selected for the target tasks, the planner starts applying these methods in order (skipping the "method selection" step). Once all target tasks have been decomposed into primitive actions, the remaining template tasks are addressed. This is done in a random order to promote variety between multiple scenarios generated using the same input.

Method Selection

When a task has to be decomposed that does not lie within the decomposition tree of the target tasks, the planner is free to select the most appropriate method. This selection is guided by two considerations. First, the planner selects the methods with the highest number of currently established preconditions. If there are multiple candidates, the system continues by considering the difference between the difficulty levels stored in the method and the difficulty level best suited for the trainee. The difficulty level of the method is computed as a weighted sum, using the weights and skill levels stored in the method. The difficulty level that best suits the trainee is computed by first requesting the relevant skill levels from the Student Model. If any of the subtasks equals the target task of the scenario, the requested difficulty level specified in the input is used instead of the skill level from the Student Model. Using the skill levels from the Student Model (or input) and the weights stored in the method the weighted sum is computed. This sum represents the difficulty level that best suits the trainee. The absolute difference between the difficulty level of the method and the difficulty level that suits the trainee is indicative for the selection process. The Action Planner selects the method with the best match (i.e. with the smallest difference) with ties being broken randomly.

Ensure Preconditions

Once a method has been selected for application, the Action Planner tries to fulfil any open preconditions. These preconditions specify services that have to be present in the game world. The Action Planner sends these requests to the Object Selector. Apart from the desired service, the Object Selector also requires a desired difficulty level as input. For this value the Action Planner reuses the difficulty level that it determined to be most suitable for the trainee during the “method selection” step.

Decomposition

The next step of the Action Planner is to actually apply the selected method and replace the high-level task with its subtasks. In this step all constraints associated with the task are transferred to the subtasks. If one or more of the subtasks are primitive tasks, the planner selects and applies the corresponding operators. In our framework, operators are similar to the operators defined for classical planning with the exception that the preconditions can also identify specific services that have to be offered in the game world. These service prerequisites are fulfilled by the Object Selector in a way comparable to the application of a method. Any other preconditions are realised through the use of plan-space planning techniques. For every missing precondition the planner selects an operator that introduces a causal link fulfilling the precondition. Additional constraints have to be added to ensure that no other operators are introduced that can threaten this link. This plan-space process actually plans actions not for the trainee, but for the virtual agents. Therefore, if an action is planned there also has to be an agent selected that will perform the action. This job is deferred to the Object Selector. The Object Selector first checks the agents already present in the game world. If there are no agents present or none of the agents can perform the action, a new agent is added to the world and instructed to achieve the goal for which the action was selected. It is important to notice that the agent is not instructed to perform the action, but to achieve the goal that caused the Action Planner to select the action. This will allow the agents to accomplish this goal in their own way using planning mechanisms such as described in [1]. The action has to be planned, however, because the system needs to know if the agents will require any additional services that have to be added to the game world.

Once the plan contains no more high-level tasks the planning process is finished. The planner returns the plan which, together with the objects selected by the Smart Objects, and the agent goals determined in the plan-space planning steps, form the output of the Scenario Generator.

4.4 Object Selector

The underlying idea for our approach is that what the trainee experiences during the scenario is determined by the game world and the behaviour of the objects and agents within [26]. Our system therefore constructs this game world in such a way that the trainee will be required to practise the target tasks. The Object Selector is responsible for choosing the appropriate content. In order to make such a choice the Object Selector must be able to reason about the available objects and the services they offer. Therefore, the system employs Smart Objects. Agents, representing both human and non-human entities, are also considered Smart Objects since they can offer services. The following sections first detail the Smart Objects and the semantic knowledge they carry, and then explain the selection process.

4.4.1 Smart Objects

This approach builds upon the concept of Smart Objects as introduced by Kallmann and Thalmann (1998) [21]. Similar to the work of Lopes and Bidarra (2011) [26] the notion of Smart Objects is extended to not only contain animation intelligence, but to also embed information related to the interaction possibilities an object can offer the trainee. To this purpose additional data is embedded into the Smart Objects regarding the service implementations they offer and their fit into specific settings.

A *service implementation* is a data structure that contains concrete information on how an object can fulfil a specific service. This includes data on how an agent should use the object to obtain the specific service, the preconditions for this use, the constraints on its use, and the range of difficulty levels at which this service can be offered. Data relating to how an object should be used include the traditional animation intelligence of Smart Objects. It specifies the steps the agent has to perform, and the animations associated with each step. The preconditions of the service implementation specify requirements on the game world that have to be met before the object can offer a specific service. There are two types of preconditions. Some conditions can be fulfilled without introducing new requirements on the game world. For example, a gas stove has to be lit before it can offer heat. On the other hand, some preconditions might require the availability of other services, such as an agent requiring a hot object before it can offer the service 'burn wound'. The constraints on the use of a service implementation restrict the services that can be offered by one instance of that object. An example constraint would be that a human agent cannot simultaneously offer the services of being conscious and being unconscious. Finally the difficulty range of a service implementation is specified with two numbers between 0 and 1. This difficulty range has three interpretations. First, some objects are simply more difficult in use than other objects. For example, using an electric screwdriver is easier than using a penny to tighten a screw. Secondly, a service implementation can be more or less difficult because it is more or less obvious that the object can be used for this service. The same example can be applied here. A penny can be used as screwdriver (depending on the screw), but this will require an additional creative thinking step on the part of the trainee. Both these difficulty levels are static properties of a service implementation. The third interpretation allows for a more dynamic difficulty level and accounts for the use of a range of difficulty levels instead of a single value. Some objects might be able to offer the same service at different difficulty levels. A small kitchen fire can offer the service 'danger' at an easy level or it can become a raging inferno and offer the same service at a more difficult level. Although not directly relevant for the Scenario Generator, this difficulty range should also allow the encompassing AEG to adapt the difficulty of the scenario at runtime.

Apart from the service implementations it offers, a Smart Object also contains information that indicates how well the object would fit in a specific setting. A comfortable couch is a common object in a house, but is less likely to be found in a park.

4.4.2 Selection Process

When the Object Selector is requested to fulfil a specific service it will first check the current state of the game world. If the current state already offers the service or if there is an object present that could offer the service, the Object Selector checks the constraints of the associated service implementations. If the new service request can be unified with these constraints, the Object Selector is done, otherwise it will try to find a new object to add to the game world. This preference for the reuse of existing services and objects has been added to enhance the consistency and authenticity of the world. Consider the situation where the game world already contains a fire (because one of the tasks of the trainee is to extinguish a fire). Next, the Object Selector is requested to find an object that can offer the service 'heat' for a victim to burn himself on. Although it would be possible, the victim would have to be very unlucky indeed to burn himself on an object completely unrelated to the fire in question. However, in some applications the occurrence of multiple service instantiations might be desired and more complex probabilistic selections models could be employed.

If the Object Selector has decided to add a new object to the game world, it starts by determining a candidate set of all objects in its knowledge base that could possibly offer the requested service. The objects in this set are sorted first on their suitability to the setting specified for the scenario, and secondly on how close they can approach the requested difficulty level. All objects that are considered equal with respect to these rankings are shuffled randomly. Next, the Object Selector considers the first candidate and checks if its preconditions are met. If the preconditions specify services that are not yet offered by the game world, the Object Selector will recursively try and find a new object that can offer this service. Other open conditions are left for the Action Planner to resolve. If a precondition cannot be established, the second candidate is considered and this process continues until a suitable candidate has been found. Once an object has been found for which all preconditions are fulfilled, the Object Selector checks the constraints. If the constraints can be unified with the current state of the game world, the object is selected and the process ends. If no suitable object can be found the process fails.

When the Object Selector has chosen a new object to place in the game world, the object is added to the scenario plan. An object is annotated with the difficulty level at which it is to offer its services. This value is as close to the required difficulty as the difficulty range of the service implementation allows.

4.5 Scenario Requirements

The process of scenario generation has been designed in such a way that the resulting scenarios meet the requirements set to ensure effective training.

Focused on the learning objective: The AEG translates the learning objectives into tasks the trainee has to perform, and these tasks form the basis of the training. This ensures that the scenarios offer the trainee the opportunity to practise the learning objectives.

Adapted to the competency level of the trainee: This adaptation is accomplished in two ways. Firstly the choice of the HTN methods (see Section 3.1.4) to use for decomposing high level tasks into concrete actions is influenced by the desired difficulty level and the current competency level of the trainee. Secondly, the choice for, and the parameters given to, a Smart Object are dependent on the difficulty level.

Complete exercises: Using a scenario template as a backbone when constructing the storyline allows the process to generate scenarios that encompass all aspects of a training exercise.

Authentic: HTN planning has the great advantage over other planning techniques in that it respects the user's intentions. Only solutions that follow the user-specified decomposition schemas are considered. This quality aids in ensuring the authenticity of the training scenarios. Moreover, the system takes care to select objects and methods that best fit the current state of the world, ensuring a coherent story line.

Varied: The use of Smart Objects provides the required variety in the scenario generation process. Because the system reasons with services instead of concrete objects it is very easy to add additional objects that enhance the variety between the scenarios.

Chapter 5

Implementation

To test the design of the Scenario Generator discussed in the previous chapter, I have implemented a prototype that generates scenarios for First Aid training. This chapter discusses this prototype and the choices that were made during the implementation. The first section considers the training domain and the second section discusses the prototype.

5.1 Training Domain: First Aid

For this prototype the training domain of First Aid was selected. It is a field where SBT in the form of role play is of high importance. People need to be able to experience and practise with life-like situations. Considering time and resource constraints, First Aid also has the advantage that it is a relatively simple domain. There are clearly defined rules and procedures that should be followed in all situations. There are no complex representations required to model the state of the world, no uncertain outcomes, and there is no need for an explicit time representation. Moreover, there is extensive literature available specifically written to be understandable to a layman. This last quality allowed me to construct an initial knowledge base without the aid of a domain expert. Finally, the most important events that happen in a First Aid scenario are those that consitute the accident. All other events are usually direct results of the behaviour of the trainee (for example whether the victim's condition will deteriorate or improve). At the beginning of the scenario the trainee does not yet have the opportunity to disrupt the planned course of events, making a predefined scenario plan more useful than in highly interactive situations.

To limit the time necessary for developing a knowledge base, the considered situations and tasks are limited to burn scenarios. Since burns can occur in many situations and by many different causes this limitation still allows the system to demonstrate its potential.

5.2 Prototype

The prototype discribed in this section is an implementation of the framework discussed in the previous chapter. However, some adaptations had to be made regarding the design to account for time and resource constraints. First of all, the planner employed by the prototype is a rudimentary HTN planner instead of a hybrid HTN planner. This choice was made because implementing a hybrid HTN planner from scratch would be too time-consuming and no suitable implementations could be found to build on. The only hybrid HTN planner that is freely available is the HyHTN planner [1]([30]). However, this implementation is written in poorly documented Prolog, making it very difficult to extend and thereby unsuitable for this project. The main consequence of this change is that the system only reasons about services and objects, and does not model any other aspects of the state of the game world. For the domain of First Aid this is an acceptable

[1]http://scom.hud.ac.uk/planform/resource/HyHTN/htncode.pl

limitation, because the only things that really matter during a training session are the agents, their behaviour, and the availability of tools that the trainee might require to provide First Aid. A related simplification, is the omission of ordering constraints between the actions of the virtual agents (i.e. the events) and the actions of the trainee. Such constraints were not necessary for the current domain since, as mentioned above, in First Aid scenarios all the required events happen before the trainee comes into action. A final alteration is the use of the difficulty level. Because there is no Student Model available, the prototype uses a simpler difficulty concept. Instead of a value between 0 and 1 for every learning objective, the trainee is modelled as being either a beginning or an advanced student. This simplification is not considered to be a serious limitation for this prototype, because First Aid instructors likewise do not keep track of the skills of the trainees, and only discern between a beginner course and an advanced course. The implications of these simplifications for the extendibility of the prototype to other domains are considered in the discussion.

The remainder of this section first details the in- and output of the prototype, and then considers some general implementation issues. It concludes with a discussion of the implementation of the scenario generation process.

5.2.1 Input

This section details the parameters the prototype requires as input.

Domain Knowledge

The first input parameter is the path to an XML file containing the domain knowledge. To structure this knowledge, and to allow for easy extension and reuse across different training domains, an upper-ontology is developed. This ontology describes the concepts that the Scenario Generator uses to reason about a domain. An overview of the ontology is shown in Figure 5.1. The ontology is designed using the Protégé open source ontology editor. Most concepts have already been discussed during the specification of the design, but some additional classes are added to facilitate hierachically ordered Smart Object classes. Below, all concepts are listed along with a brief explanation.

- *Task*: represents a non-primitive task that can be decomposed into other tasks or atom tasks according to a decomposition schema. A task can have one or more applicable decomposition schemas.
- *Atom Task*: a primitive task (i.e. action). It can have zero or more preconditions that specify services which have to be offered by the game world before this task can be performed.
- *Decomposition Schema*: represents an HTN method. It defines how to decompose a specific task into one or more subtasks that can be either tasks or atom tasks. A decomposition schema can have zero or more preconditions in the form of services or specific objects. Because of the different use of the difficulty level, a decomposition schema has a difficulty range rather than a vector of difficulty levels and associated weights. This range indicates the minimum and maximum difficulty that decomposing the task can offer.
- *Scenario Template*: an ordered list consisting of tasks and/or atom tasks that together constitute a complete exercise.
- *Service*: indicates an interaction possibility that an object can offer. To ease the design process services can be defined hierarchically using inheritance relationships. For example, the service ‘chemical burn’ inherits from ‘burn’ which in turn has ‘injury’ as a parent.
- *Service Implementation*: specifies how a specific service can be offered by a specific type of object. A service implementation has a difficulty range, indicating how difficult it is to use this service, and a list of preconditions and constraints. Because the planner does not facilitate action planning, the preconditions are specified as actions rather than goals, thus the planning step can be skipped. Moreover, for every action an agent type is specified that is capable of performing this action. The constraints

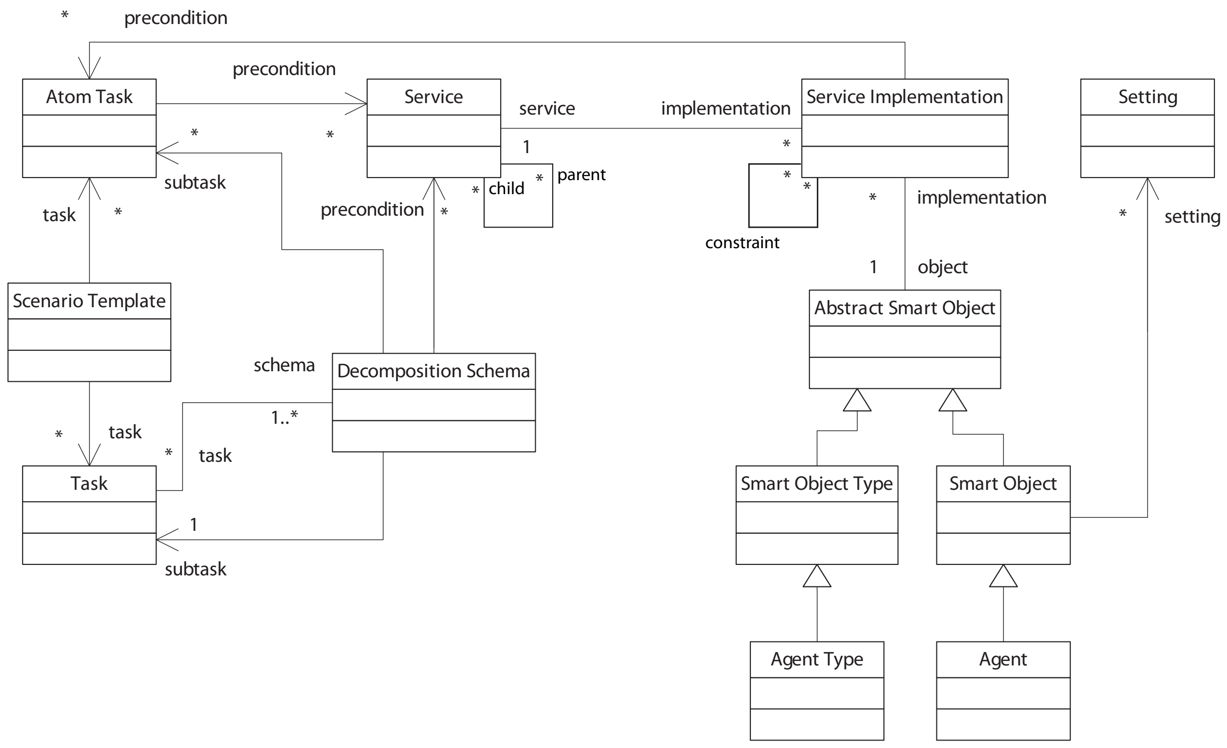

Figure 5.1: An UML diagram of the upper-ontology that structures the domain knowledge concepts.

identify other service implementations that the object might offer, but that are incompatible with the current service implementation.

- *Abstract Smart Object*: a general concept that encompasses all Smart Objects.
- *Smart Object Type*: represents a class of Smart Objects that can offer specific services. An example would be the type 'wooden object' that offers the service 'fuel'. Any Smart Object that inherits this type offers the service fuel. This hierarchical structure allows the designer to quickly add new objects without having to specify all details.
- *Smart Object*: a specific object that offers services. It also contains a list of settings in which one would be surprised to find the object, and a list of settings in which one would expect to find the object.
- *Agent Type*: this concept is similar to that of a Smart Object Type, only this type refers specifically to agents.
- *Agent*: a specific type of Smart Object that is not only able to offer services, but can also perform actions.
- *Setting*: identifies a contextual setting for the scenario.

The actual domain knowledge is input as instances of the classes defined by the ontology. For the prototype this instantiation was done based on the competency guide of "Het Oranje Kruis", the official Dutch association for First Aid which determines the teaching curriculum and is responsible for the examination and certification of trainees.

Task

This second input parameter identifies the target task to be trained. In the current implementation only one target task can be specified for a scenario. This task can be either a primitive or non-primitive task.

Difficulty Level

As explained earlier, the difficulty level can be one of two values, beginner or advanced.

Scenario Template

This input component identifies the scenario template the prototype is to use. In the current implementation there is one template available, specifying the tasks that the trainee should perform during every exercise: secure environment, calm victim, treat.

Setting

The value for this parameter identifies the setting where the scenario is to take place.

5.2.2 Output

The output of the prototype differs slightly from the scenario plan specified in the previous chapter. Because the prototype does not work with an action planner and because the results are not sent to autonomous agents, the high-level agent goals are replaced with descriptions of agent behaviour. Table 5.1 shows an example of the output of the prototype. As can be seen in the table the output consists of five lists. The first list specifies the planning for the trainee, it also indicates which objects he is expected to use to give the viewer a better understanding of the workings of the system. The second list describes the agent behaviour that forms the story line of the scenario. The next two lists detail all objects that have to be added to the game world. They are divided into static objects and dynamic objects, where dynamic objects represent agents. The final list repeats all the objects that have to be placed in the game world, but now details the services they offer and the difficulty level at which they are to offer these services.

This output describes a scenario where a beginner level trainee has to perform the task 'treat burn' in a home setting. The prototype instructs the victim agent to drop a hair dryer into the water, which results in the victim receiving an electric shock. The victim is groggy and has an electrical burn which the trainee then has to treat. Because the trainee is a beginner, all services are offered at an easy level (1 for beginner, 2 for advanced).

5.2.3 General Implementation Considerations

Two general implementation issues that had to be considered were the interpretation of the domain-specific knowledge and the facilitation of backtracking options. The first challenge regarding the domain-specific knowledge was caused by the desire to keep the reasoning process of the prototype domain-independent. As a result, the prototype can only reason with the class structure as defined by the ontology and not with domain-specific classes. The domain-specific information is interpreted at runtime, and stored as instantiations of the domain-independent classes. For example, the object 'gauze bandage' is represented as an instance of the class 'SmartObject' and not as a class by itself. However, since it is possible that a domain-specific instance, such as an object or action or task, occurs more than once in the scenario plan, the need for unique identification arises. To facilitate this unique naming demand, an additional abstraction layer has been added to the class structure of the prototype. It differentiates between description classes, and concrete classes. The description classes are the interpretation of the domain-specific knowledge. The

```
Task to train: treat_burn in the Home at level: 1

Planning:
turn_off_power using hair_dryer for current,
determine_grogginess using victim for grogginess,
cool_burn using victim for burn, tap for water,
bandage_wound using victim for injury, towel for bandage,

Story:
victim performs electrocute to obtain electric_burn using hair_dryer for current,
victim performs drop_in_water to obtain current using tap for water,
victim performs electrocute to obtain grogginess using hair_dryer for current,

Static Objects:
tap
hair_dryer
towel

Dynamic Objects:
victim

Services:
tap offers water at level: 1
hair_dryer offers current at level: 1
towel offers bandage at level: 1
victim offers electric_burn at level: 1
victim offers grogginess at level: 1
```

Table 5.1: An example of the system output. The output consists of the input parameters followed by five lists: *planning*, the actions the trainee is to perform; *story*, the actions of the virtual agents; *static objects*, the objects present in the game world; *dynamic objects*, the agents present in the game world; and *services*, the parameters for the services offered by the objects and agents.

concrete classes are actual actions or objects that are added to the scenario plan. They are annotated with a unique identifier and a pointer to the description class that gives them their domain-specific meaning. Continuing the example, the object 'gauze bandage' is actually represented as an instance of the description class 'SmartObjectDescription'. When a specific bandage has to be added to the scenario plan, a new instance of the concrete class 'SmartObject' is created to represent this unique bandage.

The second challenge that had to be overcome during implementation, was the facilitation of backtracking options. To allow the prototype to backtrack along certain choices, the state of the world is copied at every backtrack point. The prototype then continues its reasoning process based on this copy. Only when it is certain that it has made the right choice, the copy replaces the original. The backtrack points for the system are the choice for a decomposition schema and the choice for a service implementation.

5.2.4 Scenario Generation Process

The scenario generation process follows the steps as explained in the previous chapter. First, the target task has to be fitted within the scenario template. Then, the target task is decomposed followed by all other template tasks. Once a task has been selected for decomposition, an appropriate decomposition schema has to be selected and its preconditions ensured. Next, the decomposition schema is applied and the preconditions of any resulting atom tasks are established. The remainder of this section elaborates on the implementation of these steps.

The fitting process is accomplished by a brute force approach. The system considers all tasks from the template in order. For each task, it determines all available decomposition schemas, and stores their subtasks in a new list. When all top-level tasks have been considered, the algorithm recursively repeats the process with this new list of subtasks as input. For each (sub)task it encounters, the system stores two pointers. The first pointer references the parent task, and the second pointer represents the decomposition schema that was used to decompose this parent task into the current subtask. When the system has decomposed all tasks into atom tasks, it can look up the target task and follow the parent pointers to the template task whose decomposition ultimately resulted in the target task. When there are more paths available the system chooses randomly. Although this process can become inefficient for large domains, the pointer table has to be created only once and can be reused for other scenarios. Because the table stores all possible paths between the top-level tasks and atom actions, and the system is free to choose amongst these paths, the reuse of the table will not influence the variety between the scenarios generated for the same target task.

The task selection mechanism of the prototype works as follows. The prototype always starts with the decompositions of the template task to which the target task was fitted (denoted as template target task). Only when this task has been completely decomposed into atom tasks the other template tasks are considered. The system then starts with the decomposition of the task preceding the template target task, followed by the task preceding that until there are no earlier tasks. Then the system considers the task following the template target task until all tasks from the template have been completely decomposed. Thus, if there are five tasks in the template numbered 1 through 5, and task number 3 is the top-level target task, the decomposition order would be 3, 2, 1, 4, 5. This order ensures all semantic constraints between the top-level tasks are maintained. For example, if the target task is to 'treat a chemical burn', it would be inconsistent if the decomposition of an earlier template task such as 'secure the environment' would result in a fire that has to be controlled.

When a task has to be decomposed for which there is not already a decomposition schema chosen in the fitting step, the system considers the available decomposition schemas. It first sorts the schemas based on how closely their difficulty matches the desired difficulty for the scenario. All schemas that score equally on this difficulty match are shuffled randomly. The system then considers the schemas in order. For ease of implementation, the system does not select the schema with the highest number of satisfied precondition, the approach specified in the design, but selects the schema for which all preconditions can be satisfied without the addition of new objects to the game world. This approach is stricter, but in practice does not influence the results because with the current decomposition schema instances, either all or none of the preconditions are satisfied.

As a result of the decomposition schema selection procedure, the step of establishing the preconditions of a schema is only relevant for the decomposition of the template target tasks and its descendants, or when a precondition of another method can be fulfilled without adding a new object to the world. An example of the latter situation occurring would be when the schema requires the service 'smoke', and there is already a fire present (to offer the service 'heat'). When a service has to be offered, the system first considers all objects present in the game world. If none of these objects can offer the requested service, it adds a new object. The object is selected based on the difficulty level match. Once an object is selected, the preconditions of the associated service implementation have to be established. As explained earlier, these preconditions are represented as atom tasks that need to be performed by the agents in the game world. The system first checks if the atom task has any preconditions of its own. If this is the case, the system recursively tries to establish these conditions. When all conditions are fulfilled, the system appoints the atom task to an agent. This assignment is based on the desired type of the agent as specified in the service implementation. If the type is the special indicator 'self', the object that offers the service implementation is an agent, and the action has to be appointed to that specific agent. Otherwise, the system ascertains whethere there is already an agent of the correct type present in the game world. If a suitable agent is found, the action is asigned to him, if not, a new agent is added.

The actual decomposition step entails replacing the selected task by its subtasks, and establishing the preconditions of any resulting atom tasks. This establishment is done using the same procedure as described in the previous step.

Chapter 6

Evaluation

The goal of this research project has been to design and evaluate a framework that automatically generates training scenarios tailored to the abilities and needs of the individual trainee. To evaluate the framework a prototype has been implemented and an experiment conducted to assess the quality of the resulting scenarios. This chapter details the experimentation process. After an introduction which explains the general setup and research questions, the methods and results are described. The chapter concludes with a discussion of the results.

6.1 Introduction

The system was designed to generate scenarios for use in Adaptive Educational Games (AEG). To ensure effective learning, these scenarios have to be tailored to the needs of the trainee. The basic idea behind the framework is to plan the actions of the trainee and the other agents in the virtual world, and to facilitate these actions by selecting the appropriate game content. The prototype assumes that the task to be trained and the competency level of the trainee are known. It uses this knowledge as input, along with a specification of the setting where the scenario has to take place and a scenario template. This template specifies an ordered sequence of high-level tasks, which together constitute a complete training scenario. The prototype combines two mechanisms to generate a scenario: HTN planning and Smart Object selection. The planner determines the actions of the trainee and the virtual agents, and the object selector facilitates these actions in the game world. The prototype first employs its rudimentary HTN planning algorithm to decompose the high-level tasks of the scenario template into low-level actions. This decomposition process is guided by the competency level and task to be trained. As soon as an action is selected, the object selector comes into play. It selects objects that can fulfil the requirements the action might impose upon the game world. For example, the action 'bandage burn' requires a burn and something that can be used to bandage the burn. The selector mechanism chooses objects that can fulfil these services taking into account that some objects are more likely to be present in specific settings than others, and that some objects might be (or can be instructed to be) more difficult to use than others. For example, the use of a gauze bandage to dress a wound is straightforward, but it is not as easy to recognise that the same result can be accomplished with a clean T-shirt. Furthermore, a victim can be a bit startled and scared or he can be hysterical. This object selection mechanism is an iterative process since some objects require specific actions before they can be used. The output of the system consists of four parts. First, the actions the trainee is expected to perform are listed along with the objects he is to use. The next list enumerates the actions of the virtual agents (i.e. the events) along with the associated objects. The third part describes all objects that have to be present in the game world. This list is split up in static and dynamic objects, with the latter being agents. The final list again depicts all the objects present in the game world and supplies additional information regarding the services they are to provide and the difficulty levels at which they provide them.

To evaluate the prototype a measure was needed to determine the quality of the generated scenarios. Based on the earlier discussion of SBT (see Section 2.3) three important requirements were selected to measure the quality of a single training scenario. First of all, the scenario has to be suitable for the task to be trained. It has to offer the trainee the opportunity to practise this task. Secondly, the scenario has to be suitable for the competency level of the trainee. The scenario has to be challenging, but not overwhelming. Finally, the scenario has to be authentic. It is important that the trainee feels that the scenario is relevant and depicts a real world situation. Using these requirements, the quality of the generated scenarios can be determined by comparing them to scenarios authored by human experts. The expert scenarios serve as an example of the desired level of quality. Because of the exploratory nature of this project, it was anticipated that the system might not yet reach such standards. Therefore, it was decided to also compare the generated scenarios to those of human laymen. It was anticipated that the combination of these comparisions would provide a better understanding of the strengths and weaknesses of the system.

6.1.1 Research Questions and Hypotheses

The main research question is: *How do automatically generated scenarios compare to scenarios written by human laymen and experts with respect to authenticity and suitability for a specific task and trainee?*
To answer this question the following sub questions need to be answered:

- *How do automatically generated scenarios for a specific task and difficulty level compare to those written by a human experts and laymen with respect to their suitability for the task?*
- *How do automatically generated scenarios for a specific task and difficulty level compare to those written by a human experts and laymen with respect to their suitability for the competency level of the trainee?*
- *How do automatically generated scenarios for a specific task and difficulty level compare to those written by a human experts and laymen with respect to authenticity?*

I hypothesise that the system will perform better than laymen but not as good as experts on all three requirements.

6.1.2 Domain Choice

To ensure that scenarios of different sources could be compared fairly, the domain for the experiment was restricted to that of First Aid in burn-related incidents. The choice for First Aid scenarios was made during implementation of the prototype (see Section 5.1). The restriction to 'burn situations' was added for two reasons. Firstly, to limit the required knowledge of the system, and secondly, because laymen have enough common sense knowledge regarding this topic to write meaningful scenarios. The restriction to 'burn situations' imposes all scenarios independent of the target task, to take place in situations related to burns. For tasks such as 'treat burn' this seems straightforward, but it also applies to more general First Aid tasks such as 'ensure safety' that could be relevant in any number of situations.

To negate any confounding influences, the task to be trained, the competency level of the trainee and the fictional location of the scenario, were counter balanced during the experiment. Three tasks were selected (treat burn, ensure ABC and calm victim) and combined with two difficulty levels (beginner and advanced). Four locations were chosen (in a park, at home, at a restaurant and in a laboratory). To keep the number of scenarios within acceptable limits, each task/difficulty combination was set in only two locations. This procedure resulted in 12 unique scenarios sprecifications see Table 6.1.

	Beginner				Advanced			
	Park	Home	Restaurant	Laboratory	Park	Home	Restaurant	Laboratory
Treat burn	x	x					x	x
Ensure ABC	x			x		x	x	
Calm victim			x	x	x	x		

Table 6.1: This table identifies the 12 selected task/difficulty/location combinations used in the experiment.

6.2 Method

6.2.1 Participants

This experiment required two types of participants: evaluators and scenario writers (both expert and layman).

Evaluators

Five experienced First Aid instructors participated in this study. They were contacted through their respective First Aid affiliations. Although all instructors were recruited from different affiliations, some had worked together on previous occasions.

Expert Writers

Five First Aid instructors were asked to write scenarios. All instructors were recruited through their respective affiliations. By contacting affiliations located far away from those where the evaluators were recruited, interference between these two groups was avoided.

Layman Writers

Four young adults between the ages of 24 and 30 years were recruited as layman writers. All were university students or graduates but from different areas of study. They had no experience giving or receiving First Aid training.

6.2.2 Research Design

Based on the research questions, three dependent variables were identified: suitability of the scenario in regard to the task to be trained (*task suitability*); suitability of the scenario in regard to compentency level of the trainee (*competency suitability*) and *authenticity* of the scenario. The independent variable was the *source* of the scenario. The *source* could be one of three options: expert, layman or system, resulting in three conditions.

The experiment was setup as a within-subjects design with all instructors assessing all scenarios from all conditions. It was anticipated that this design would minimise the error variance associated with individual instructors and reduce the number of evaluators required.

6.2.3 Materials

Since the prototype was not yet able to interact with a virtual world to create the scenarios, the experiment used textual descriptions of the scenarios. All scenario descriptions were styled in accordance with a specific format. Since there exist no standards or rules for describing First Aid scenarios, a format had to be designed specifically for this study. The design was based on some examples of scenario descriptions as given by an experienced instructor. It contained all important aspects needed for an instructor to prepare and set up a training session. Table 6.2 shows an example of a scenario description styled according to the format. First, the target task, difficulty level and location are provided. Next, the background story is described, detailing how the current situation came about. Then follows information pertaining to the behaviour and appearance of the victim(s). This includes detailed discriptions of injuries, relevant items of clothing or accessories and notitions on the mental state of the victim. Finally, all objects necessary for the enactment of the scenario are descibed. This includes objects relevant to the occurrence of the accident but also object the trainee could use in aiding the victim. It was assumed that no other relevant objects were present unless mentioned in this field.

To obtain the human-authored scenario descriptions, four laymen and five First Aid instructors (experts) were contacted (the scenario descriptions written by the fifth expert were used in the training set, see Section

Task, level and location	Task: calm victim	Level: advanced	Location: home
Background story	Henk has to declog the sink. He thinks this is an annoying job and wants it to be over with quickly. Because of his haste he spills some of the drain cleaner fluid over his arm. Even through his shirt he feels the fluid stinging his skin and he panics.		
Behaviour and appearance LOTUS-victim	Henk panics. He never realised a common household chemical could be this dangerous. It is difficult to calm him down. A large second degree burn with blisters forms where the drain cleaner touched his skin. The shirt Henk is wearing is sticking to the wound.		
Objects	A half empty bottle of drain cleaner lays in the sink. There is tap attached to the sink and there are bandages in a dresser.		

Table 6.2: Scenario description example.

```
Task to train: calm person at home at level: 2

Planning:
talk_to_person using victim for panic,
cool_burn using victim for burn, tap for water,
remove_clothing using victim for clothing_near_wound,
bandage_wound using victim for injury, gauze_bandage for bandage,

Story:
victim performs get_burn_chem to obtain chem_burn using drain_cleaner for chemical,
victim performs panic to obtain panic using victim for danger,

Static Objects:
drain_cleaner
tap
gauze_bandage

Dynamic Objects:
victim

Services:
drain_cleaner offers burning_chemical at level: 1
tap offers water at level: 1
gauze_bandage offers bandage at level: 2
victim offers chem_burn at level: 2
victim offers panic at level: 2
victim offers clothing_near_wound at level: 2
```

Table 6.3: System output example.

6.2.5). They were asked to each write three scenarios for specific task/difficulty/location combinations using the format described above. The partitioning of these combinations were the same for both groups, laymen and experts. For the laymen some additional information regarding the practice of First Aid training sessions and the specific tasks was provided.

The system was given the same input as the human authors with respect to task, difficulty level and location. However, since the system did not return a story-like description, an additional processing step was required. The output of the system was rewritten to resemble the style of the human authors. A rule set was developed that dictated how to translate the output of the system into natural language. As explained in the introduction, the system output consisted of four lists: trainee actions, story actions, objects and services. The service and object list are somewhat similar in that they both enumerate the same objects. However, the object list only devides the objects into two groups: static or dynamic, and could be ignored during the rewriting process. Futhermore, the trainee actions were not included into the scenario description since it could be assumed the instructor already knew the proper procedures. The information contained in the other two lists was distributed over the relevant scenario description boxes. The story actions formed the background story. The service list offered information on both the victim's behaviour and appearance box, and the required objects field. All services that were offered by agents were input in the behaviour and appearance box. The objects associated with the remaining services were placed in the objects field. The service list did not only specify which services an object (or agent) had to offer, but also the difficulty level at which they were offered. This information was essential for the translation of agent behaviours. Different translations were made depending on the difficulty level of the offered service. If, for example, a victim offered a burn at level 1, the translation was a large first degree burn with perhaps a small second degree burn. If, on the other hand, the victim offered a burn at level 2, the translation would be a large second degree burn with blisters. For most of the objects input into the object field the different difficulty levels did not influence the translation process because they had no 'behaviour' they could adapt. These difficulty levels were merely codes that the scenario generation system used in its internal decision process and were therefore ignored during the translation. The only information that could not be deduced from the system's output were the motivations or feelings that drove an agent. In the example of Table 6.2 the fact that Henk thought the job annoying and spilled the fluid because he was hasty, could not be found in the output of the system, but was added to make the agent sound more believable. This addition of information is assumed to be justified because in a fully functioning AEG virtual agents will be employed that are capable of inferring any required information regarding motivation and feelings to maintain a believable character. Tabel 6.3 shows the output of the system which resulted in the scenario description displayed in Tabel 6.2.

6.2.4 Measures

A custom made questionnaire was used to measure the dependent variables. For all scenarios that had to be assessed, a description of the scenario (without the target competency level) was given followed by the four questions shown in Figure 6.1. As can be seen from this figure, the evaluators were asked to give their answers to question one till three on a seven-point Likert scale. The fourth question was a multiple-choice question with additional space for elaboration in case of a negative answer.

The questionnaire was designed to not ask directly for the dependent variables *task suitability* and *competency suitability*. Instead it asked the evaluators to rate the suitability for both competency levels (beginner and advanced). The *task suitability* could then be computed as the highest score to either question. If, for example, the scenario was deemed very unsuitable for a beginner but tolerably suitable for an advanced trainee, then apparently the scenario *was* suitable for the task, although not for beginners. The *competency suitability* was computed as the score of the question matching the competency level of the design. Thus, if the scenario was designed for a beginner the *competency suitability* was defined as the answer to question one, and if a scenario was designed for an advanced trainee the *competency suitability* was defined as the answer to question two. This indirect design was chosen because it was assumed that it would be difficult for the evaluators to rate the suitability of a scenario for a specific task without being influenced by the suitability of the scenario for the trainee's competency level. It also provided more information on the changes that could be made to improve the quality of the scenario. If a beginner scenario was deemed

1. How suitable do you find the scenario for the given task and a *beginner*?

-3	-2	-1	0	1	2	3
very unsuitable	tolerably unsuitable	somewhat unsuitable	not suitable/ not unsuitable	somewhat suitable	tolerably suitable	very suitable

2. How suitable do you find the scenario for the given task and an *advanced* trainee?

-3	-2	-1	0	1	2	3
very unsuitable	tolerably unsuitable	somewhat unsuitable	not suitable/ not unsuitable	somewhat suitable	tolerably suitable	very suitable

3. How believable do you find the scenario?

-3	-2	-1	0	1	2	3
very unbelievable	tolerably unbelievable	somewhat unblievable	not believable/ not unbelievable	somewhat believable	tolerably believable	very believable

4. Would you use the scenario in your own training sessions, and if not why?

Figure 6.1: The questions as presented to the evaluators

better suitable for an advanced trainee than for a beginner, it was apparently too complicated and should be made easier. If an advanced scenario was found to be better suitable for a beginner, the scenario would have to be made more challenging.

The third question was assumed to directly measure the dependent variable *authenticity*. A different word (believable) was used, because it was anticipated that it would better convey the meaning of the dependent variable to the evaluators.

The fourth question was added as a check to determine the scenario's usefulness according to the evaluators.

6.2.5 Procedure

The experiment started with the presentation of some general information pertaining the purpose and procedures of the study. The evaluators were told they would have to rate different scenario descriptions with respect to their authenticity and their suitability for the task and trainee. They were ignorant of the different sources of the scenarios. To ensure all evaluators shared a common understanding of the meaning of the questions they were presented with a practice scenario. The evaluators were encouraged to voice their answers to the four above mentioned questions with regard to this practice scenario. To further enhance inter-observer reliability and to familiarise the evaluators with the scenario format, they were then presented with a practise set. This set contained six scenario descriptions. These scenarios were generated by both laymen and experts and concerned tasks and locations different from the test set. The evaluators were instructed to work by themselves and to not turn back to or change previous answers. After all evaluators had completed the practice set the intra-class correlation coefficient was computed. Based on the results of this computation it was decided whether the evaluators needed another opportunity to discuss their motivation and results with each other. Because of some low scores it was deemed prudent to again discuss the scenarios after which the evaluators were asked to complete a second practice set. Next, the experiment proper started and the evaluators were presented with the test questionnaire. The experiment concluded with a group discussion to obtain additional insights into the performance of the system and possibilities for improvement.

6.2.6 Analysis

An intra-class correlation analysis was performed to assess inter-rater reliability. A repeated measure ANOVA was used to determine the effects of the source of the scenario (expert, layman or system) upon the dependent variables. All analyses were performed using the IBM SPSS Statistics v.20 software package.

6.3 Results

6.3.1 Data Exploration

For the practise rounds one evaluator forgot to fill in one question resulting in a total of 239 ratings (two sets of six scenarios, with four questions per scenario and five evaluators) that were entered into the analysis. For the test set there appeared to be missing data for two scenarios for Evaluator1. Since the evaluator commented on one of these scenarios that he thought the scenario unsuitable for First Aid training a 'no' was entered as answer to question four in all subsequent analyses, resulting in seven missing values for this rater. Evaluator2 failed to answer question four for one scenario. As a result 712 (36 scenarios with four questions per scenario and five evaluators) were entered into the analysis.

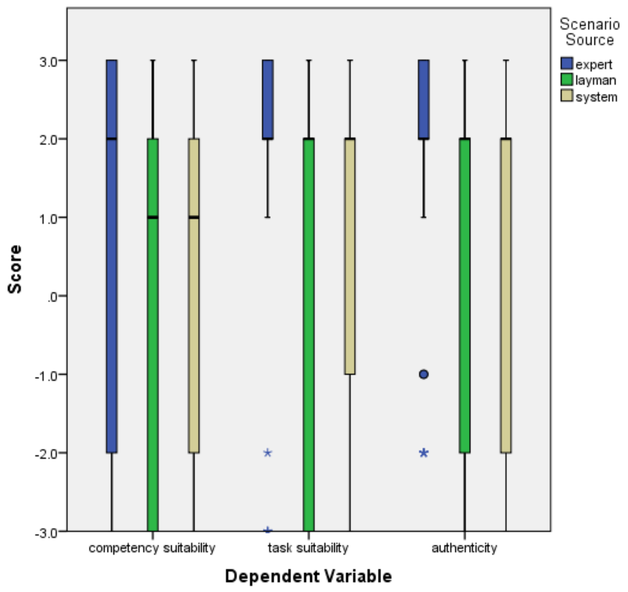

Figure 6.2: Boxplot of scores for all raters ◦) outlier, more than one boxlength away ⋆) extreme, more than two boxlengths away

Figure 6.2 shows a boxplot of the scores for the dependent variables as computed from the answers to the test set. As can be seen in the figure the median values for all sources lay close together, but there seemed to be a difference between the sources with regard to the spread of the values. With respect to the variable *competency suitability* the scores for all sources were spread out significantly. For the other two variables however, the expert scores were apparently more centered (with a few exceptions) than those of the layman and system generated scenarios. Additional comparisons between the intra-class correlations of all source/dependent variable combinations showed that this difference in spread was not caused by a lower inter-rater reliability for the latter two sources. The spread therefore suggests a larger variety between the quality of the layman and system generated scenarios compared to the expert scenarios. Table 6.4 shows the mean values and standard deviations of the test set. From this table it can be seen that for all variables the expert scenarios had the highest mean scores followed by the system generated scenarios, closely followed by the layman scenarios. In accordance with the boxplot, the table also shows large standard deviations for all sources, with smaller values for the expert scenarios regarding the *task suitability* and *authenticity*.

Regarding the additional question whether instructors would use the scenario for their own training sessions more than half of the instructors would use 10 out of 12 expert scenarios, 6 out of 12 layman scenarios and 6 out of 12 system generated scenarios.

	Scenario Source		
Dependent Variable	expert	layman	system
competency suitability	0.867 (2.2434)	0.000 (2.3915)	0.000 (2.2699)
task suitability	1.933 (1.4126)	0.414 (2.4064)	0.900 (2.1605)
authenticity	1.733 (1.4829)	0.448 (2.2098)	0.550 (2.1347)
overall	1.511 (1.8047)	0.287 (2.3330)	0.483 (2.2084)

Table 6.4: Data exploration: mean scores (and standard deviations) over all raters

6.3.2 Inter-rater Reliability

Intra-class correlation coefficients have been computed using the two-way random model based on the guidelines given in ([41]). Details can be found in Table 6.5. The overall coefficient for the questionnaire was 0.732 (p <.001). An inter-rater agreement between 0.60 and 0.79 is generally considered substantial ([25]) therefore the data were considered appropriate for further analysis.

Session	Question no.	ICC coefficient	p-value
Practice 1	1	0.893	<0.001
	2	-0.261	0.567
	3	0.616	0.057
	4	0.417	0.196
	all	0.801	<0.001
Practice 2	1	0.910	<0.001
	2	0.840	0.01
	3	0.991	<0.001
	4	1.0	.
	all	0.923	<0.001
Experiment	1	0.812	<0.001
	2	0.646	<0.001
	3	0.692	<0.001
	4	0.720	<0.001
	all	0.732	<0.001

Table 6.5: The intra-class correlation coefficients

6.3.3 Missing values

Because of the small number of evaluators, listwise deletion was considered undesirable. Little's MCAR test showed the missing values to be completely random and the percentage of missing values was less than 2% (6 out of 540). The missing values have been imputed using the Expectation Maximization algorithm. Multiple imputation was considered as an alternative but for ease of further analysis and based on the observation that there is no significant difference between both methods when the percentage of missing values is small ([15]), it was not pursued.

6.3.4 Repeated Measure Analysis

A repeated measure one-way ANOVA was used to analyse the possible effects of scenario source (expert, layman or system) on *task suitability*, *competency suitability* and *authenticity*. Some controversy exists considering the use of ANOVA (or any other parametric test) in combination with a Likert scale, since the data is officially of ordinal measurement level. However, it was felt that with the current design the step size between the items can be considered equal and the data can be said to be of interval level, justifying the use of ANOVA ([35]). The results of the ANOVA are shown in Table 6.6.

Dependent Variable	F	p-value	effect size	power
competency suitability	3.328	0.089	0.454	0.456
task suitability	6.367*	0.022	0.614	0.749
authenticity	6.145*	0.024	0.606	0.733

Table 6.6: Results of the repeated measure analysis *) p<0.05

Mauchly's test of sphericity showed that no correction was required. The repeated measures ANOVA determined that there is no statistically significant difference between the mean scores of the sources for *competency suitability* ($F(2,8) = 3.328; p = 0.089$), but there are significant differences between the sources for *task suitability* ($F(2,8) = 6.367; p = 0.022$) and *authenticity* ($F(2,8) = 6.145; p = 0.024$). However, post hoc tests using the Bonferroni correction procedure revealed that there were no significant differences between the sources once they were compared one to one. The results can be found in Tabel 6.7.

Dependent Variable	(I)Source	(J)Source	Mean Difference (I-J)	Corrected p-value
	expert	layman	0.896	0.310
competency suitability	expert	system	0.867	0.247
	system	layman	0.029	1.000
	expert	layman	1.525	0.101
task suitability	expert	system	1.033	0.223
	system	layman	0.492	0.839
	expert	layman	1.304	0.148
authenticity	expert	system	1.183	0.072
	system	layman	0.121	1.000

Table 6.7: Results of post hoc test with Bonferroni correction

6.4 Discussion of the Results

The goal of this experiment was to ascertain how the quality of the scenarios generated by the system compared to the quality of scenarios written by human laymen and experts. Three requirements were identified to measure scenario quality: *task suitability*, *complexity suitability*, and *authenticity*. It was hypothesised that for all three variables the expert scenarios would score best, followed by the system generated scenarios, and finally the laymen scenarios. The results follow the hypothesised trend. However, the differences between the sources are not significant for any of the requirements. This suggests that either there are no significant differences, or the current experiment failed to identify them. Although this first situation would bode well for the performance of the prototype, it seems highly unlikely that there would be no difference between the quality of scenarios created by experts and those written by laymen. The second situation therefore, seems more credible. Further experimentation will be required to obtain conclusive answers. Based on the current results, three points for consideration regarding the experimental design have been identified.

First of all, the selection of the scenario setting should be considered. In the current experiment, the settings of the scenarios were purposefully varied to show that the prototype was capable of generating scenarios for various situations. However, these predetermined settings might have restricted the human authors too much. Especially the expert authors might have been forced to write scenarios they would normally not have considered, lowering the average scores of the expert scenarios. One expert author indicated that it had been very difficult to come up with a proper scenario for the 'Park' setting, and that the ‘Laboratory’ setting was too specific and unsuitable for the general trainee population. For future experiments it might therefore be advisable to leave the choice of the setting to the authors. The prototype could then choose a setting either randomly or according to pre-programmed guidelines. In this way the human authors would be free to choose their own settings and write scenarios to the best of their abilities. As a result future experiments may show the quality scores for the human authored scenarios to be spread out less and the differences between the sources to be more pronounced.

A second point for consideration, is the format of the scenario description template, especially in combination with the training domain of First Aid. For the experiment, the domain of First Aid, and specifically the limitation to burn-related incidents, was chosen because of the every-day nature of such situations. It was assumed that this would allow a layman to develop meaningful training scenarios, because even without having knowledge of the correct procedures associated with the task, he can envision a situation that would require the execution of the task. However, combined with the current scenario description format that only describes the initial state of the scenario and contains almost no information on the expected course of events, there is little opportunity for the experts to show their expertise. The evaluators, being domain experts themselves, might make assumptions on how the scenario is expected to unfold. These assumptions might be in accordance with the vision of an expert scenario author, but might never even have been considered by a layman author. For future experiments it might therefore be advisable to extend the scenario description to include the flow of the scenario as envisioned by the author. The flow could be described in terms of expected trainee/agent interaction moments. An example would be: if the trainee does not tell the victim to sit down and breath deeply, the vicitm will faint. Because the prototype places the responsibility for the implementation of the trainee/agent interactions with the agents, such an extension of the scenario description template would also requires an extension of the prototype. The prototype would have to be enhanced with an agent module enabling it to ask the agents for their specific implementations of the planned trainee/agent interaction moments.

Finally, due to limited time and resources, the domain knowledge of the prototype was not generated or verified by domain experts. In the group discussion following the experiment it became apparant that faulty domain knowledge had caused the evaluators to rate some of the system generated scenarios as ‘very unbelievable’. Moreover, the evaluators also indicated that the suitability of a scenario for a specific trainee could not be rated totally independent from believability. Especially in First Aid, a scenario that is not authentic might be confusing to the trainee because victims show improbable symptoms, making the scenario unsuitable for both beginner and advanced trainees. This observation was confirmed by moderate possitive correlations between the answers to question 1 and 3 ($r = .513$ with $p < 0.001$) and 2 and 3 ($r = .430$ with $p < 0.001$). These results show the importance of involving domain experts during the construction of the knowledge base. Future experiments that include a knowledge base created, or at least verified by, a domain expert might show more pronounced differences between the layman and system generated scenarios.

Although this experiment has not produced any conclusive results on the quality of the system generated scenarios, it did show that experienced instructors would at least use some of these scenarios in their own training sessions. Future research is required to identify where and how the system generated scenarios differ from expert scenarios, and how these difference can be overcome. For now, it looks like the combination of automated planning techniques and Smart Objects is capable of generating quality scenarios and deserves further research efforts.

Chapter 7

Discussion

The aim of this project has been to explore whether it is possible to combine automated planning techniques with the concept of Smart Objects to generate effective training scenarios for use in serious games. For this purpose I have designed, implemented, and tested a system that combines these techniques. The experiment that compared the quality of scenarios generated by this system to that of scenarios generated by human experts and laymen, showed no conclusive results. However, it does suggest that the proposed combination of techniques is a viable option and warrants further research. This chapter first places the framework I have designed in context with other approaches to scenario generation. Then directions for further research are suggested based on the results of the experiment and inspired by the related work.

7.1 Related Work

This section serves to put the proposed framework into context with current research into scenario generation. Scenario generation has become a growing subject of interest in the academic community during the last few years. Earlier work focuses mostly on how to adapt scenarios at run time, or how to guide the assembly of complete scenarios from smaller human-scripted scenario pieces (such as [4] and [18]). Recently, some efforts have been made towards developing scenario generation systems, but no generally accepted solution currently exists. Below I discuss a few approaches that have inspired or could possibly enhance my work.

As mentioned in the Introduction, the idea for the proposed framework came from the work of Lopes and Bidarra [27]. They argue for the integration of scenario generation and content generation techniques to offer trainees individualised experiences. In related work, the same authors elaborate on the design of such an approach, and present the procedural generation framework for personalised game worlds, as briefly discussed in Section 3.2 [26]. Although this framework is not concerned with the actual content generation process, it does determine the nature of the content that has to be generated to realise a specific scenario. The framework considers how specific objects in the game world can affect player experience. For this purpose it employs Smart Objects annotated with the experience they can offer a player when placed in the game world. The framework acknowledges that different players might have different reactions to the same game world. Therefore, object-experience couplings are made dependent on the player type. The generation process now becomes an issue of selecting the right objects based on a player type and desired experience. These semantically enhanced objects can then be used to guide the content generation process. An example of this use of semantics can be found in declarative modelling techniques, as proposed in [42]. In my framework, experiences are translated into tasks. The added value of my approach is then, that it not only considers the tasks that directly follow from the training objective, but considers all tasks that the trainee needs to perform during a complete exercise. By applying HTN techniques, these tasks are decomposed into a coherent and complete set of trainee actions and associated events. These actions and events are then used to select appropriate game world content, similar to Lopes and Bidarra's approach. Such fine-grained control over the actions of the player offers the generator more options to diversify and personalise the trainee's experience, and to ensure a consistent storyline. Moreover, an AEG can use this action sequence to provide guidance to the trainee during and after training.

Another scenario generation technique that explicitly reasons about the game world is proposed by Martin et al. (2010) [29]. This technique differs from my approach in that it uses procedural modelling techniques to create game content, and to satisfy the preconditions that the learning objectives might pose on a game world. The system starts by selecting a minimal scenario, taking into account the training objectives and the desired competency level specified as input. This scenario is then extended with augmentations to support the training objectives, such as the addition of an extra target. Next, events or mini-scenarios can be added to increase the complexity level of the scenario. Together, these choices constitute the conceptual scenario. However, any of these components might have open requirements that still have to be set. This is where the procedural modelling techniques come into play. Martin et al. employ Functional L-Systems to establish these open requirements. Functional L-Systems are a specific type of shape grammar. The general use of a shape grammar is to define the replacement of lower-detail items as well as rules to add or change shapes. Martin et al. use this technique to fulfil any requirements and place the objects in the world. It allows them to call external scripts to handle specific demands, such as determining the correct location of a target. However, similar to the framework proposed by Lopes et al. this system does not consider the actual actions of the trainee, nor the additional requirements these may place on the game world. On the other hand, the use of Functional L-System to determine the specification of the selected content might be an interesting addition to our approach, worthy of further investigation.

Hullett and Mateas (2009) propose yet another scenario generation system that is most notable in this context because of its apparent similarities to the approach presented in this work [19]. However, in their system, scenario generation is equivalent to world content specification, and does not at all consider the events that are expected to occur during the scenario. The system is designed specifically for the domain of emergency rescue training. Trainees have to practise analysing and entering collapsed structures to rescue trapped victims. In this domain it is important that the damage to the structure is physically consistent and that elements like fire and flooding spread from sources in a realistic manner. To ensure this consistency Hullett and Mateas employ HTN planning techniques to propagate damage throughout the structure. Their planner's domain model is based on simplified physics rules with operators representing deformations that can be applied to the intact structure. The HTN planner ensures that the structure contains the elements necessary to train for the pedagogical goals, whilst maintaining consistency between the damaged sections of the structure. If, for example, a fire is required in a specific room but that room has no fire-source, the planner tries to start a fire in an adjacent room and spread it from there. Although this approach incorporates a HTN planner similar to the framework presented in this work, the use of the planner is different. Where my approach employs HTN planning to ensure coherence between the tasks planned for the trainee and consequently the storyline of the scenario, their system employs HTN techniques to consistently transform an intact structure into a collapsed structure, creating an environment that allows the trainee to practise the learning goal. Based on the success of their system, it might be interesting to see if the functionalities of the HTN planner in my framework can be extended to ensure the consistent integration of the placement of objects in the game world.

As mentioned in the Introduction, a second source of inspiration has been the work of Niehaus and Riedl ([34]), even though this focuses more on scenario adaptation than generation. The proposed system personalises a predefined scenario that is defined as a sequence of high-level tasks for the trainee. It can remove or change tasks that the trainee is already familiar with, and add tasks that need additional attention. A hybrid HTN planner is employed to decompose the high-level tasks into concrete actions, and fill any missing preconditions. This system, however, does not consider the realisation of the scenario within the game world. The focus is on the storyline of the scenario. In this aspect is can be seen as a cousin of automatic narrative generation techniques. Narrative generation systems can often be classified as using one of two techniques. Simulation-based narrative systems focus on selecting an initial situation, including characters and a world context. These systems often employ autonomous virtual agents to represent the virtual characters. The system steers the behaviour of these characters as the story unfolds to achieve its desired goals (see for example [2]). Deliberative narrative systems, on the other hand, solve the generation problem using a centralised reasoning algorithm (often a planner) that determines the optimal actions for all game world characters ([39]). Most of the techniques developed for narrative generation do not provide the necessary control or trainee adaptivity capabilities, to be applicable to scenario generation. They can, however, inform and enhance current scenario generation frameworks to focus more on character believability and the creation of a dramatic arc.

The last two approaches to scenario generation that I want to discuss are those of Grois et al. (1998) ([17]) and Zook et al. (2012) ([50]). Although they focus only on determining the events that make up the scenario, their approach could easily be extended to select appropriate content using for example a Smart Object selection mechanism to facilitate the chosen events. Grois et al. (2008) recognise that the events that lead the trainee to practise specific skills are causally connected to the training objectives. But, where my system uses plan-space planning techniques to reason about these causalities, Grois et al. employ a probabilistic network. The training objectives are written in the form of constraints or observations within the network. Using specific search techniques, probable paths through the network are determined that lead from these constraints to root events that might have caused these observations. In a very recent and inspirational work Zook et al. (2012)([50]) argue that planner-based approaches marginalise the importance of author intent, are difficult to employ in ill-defined domains, require a priori knowledge on the initial and goal situation, and do not allow for global evaluation of scenario quality during the planning process. For these reasons the authors suggest the use of genetic algorithms. Based on the training objectives, a set of required events is determined that forms the basis for all candidate scenarios. A candidate set is generated by adding optional events to the baseline scenarios. Then, a genetic algorithm is applied that iteratively changes (and hopefully improves) the candidates. Because the genetic algorithm does not take causality between the events under consideration there may be steps missing between the events. A trainee might be expected to be in one place to perform some action, and then suddenly be somewhere else for the next. Therefore, an additional post-processing step is required that employs plan-space planning techniques to fill the missing steps and add causal links. The authors argue that their genetic approach allows for more variety in the resulting scenarios. Both these approaches are fundamentally different from the approach presented in this thesis and it might be interesting to see how they compare in terms of efficiency, quality, and variety.

To summarise, Table 7.1 provides a comparison of the approaches discussed above regarding their abilities to: return concrete trainee actions; select events that constitute the scenario; select content for the game world; and be configurable to different domains. From this table it can be seen that the framework I propose is unique in its combination of techniques that allows it to 1) explicitly consider all the trainee actions, and 2) select the appropriate content to facilitate these actions. I believe that the former feature provides the framework with extra possibilities to personalise and diversify the requested scenarios. Moreover, it facilitates the encompassing AEG in guiding the trainee during and after training. The latter feature offers a more obvious advantage in that it facilitates the automatic generation of content. In game environments that are evermore increasing in realism and complexity, automated techniques are quickly becoming essential.

7.2 Direction for Further Research

This project has been of an exploratory nature, with time, resources, and domain expertise being scarce. Although no conclusive results have been found yet, further research into the use of automated planning techniques and Smart Object for automatic scenario generation seems warranted. Section 6.4 already suggests some improvements regarding the evaluation process. This section considers some extensions to the framework as a whole and general directions for further research.

7.2.1 Extendibility of the Prototype to Other Training Domains

The framework is designed to be domain-configurable, and should be easily extendible to other training domains given the appropriate world knowledge. However, during the implementation of the prototype, some simplifications had to be made that result in limitations for the extendibility of the system to other domains.

The first simplification I made was the use of an HTN planner instead of a hybrid HTN action planner.

	Trainee Action	Event Selection	Content Selection	Domain Configurable
Lopes and Bidarra	No	Through inclusion of appropriate Smart Objects	Smart Object selector	Yes
Martin et al.	No	Based on scenario complexity considerations	Procedural content generation algorithms	Yes
Hullett and Mateas	No	No	Through HTN planning	No
Niehaus and Riedl	Hybrid HTN planner	No	No	Yes
Grois et al.	No	Using probabilistic network	No	Yes
Zook et al.	No	Using genetic algorithm	No	Yes
My framework	Hybrid HTN planner	Through inclusion of appropriate Smart Objects	Smart Object selector	Yes

Table 7.1: Comparison of the properties of different approaches to automatic scenario generation

The absence of the action planner makes it difficult to specify the preconditions of the service implementations. Instead of literals that have to hold, preconditions are now specified as specific actions that need to be executed by an agent. This places a burden on the domain knowledge engineer because he has to carefully consider action sequences that achieve the desired result. In the current knowledge base for First Aid these action sequences consisted of one action only. For situations that require more complex sequences, however, this requirement is highly undesirable. Moreover, the concrete specification of action sequences reduces the freedom of the virtual agents, which in turn diminishes the variation in the resulting scenarios.

A second, related simplification is the omission of precedence constraints between the actions of the trainee and the events (i.e. the actions of the virtual agents). Although not a problem for First Aid, such constraints are necessary in complex domains where events have to occur in the middle of the scenario.

Although these two simplifications limit the extendibility of the current prototype to other training domains, they can be overcome by implementing a fully functional hybrid HTN planner. However, during implementation, another limitation was discovered that cannot be solved by a hybrid HTN planner. The service implementations defined for the framework have preconditions in the form of goals, which in the current prototype are represented as actions. However, whether these preconditions are formulated as actions or goals, in both cases they need to be assigned to an agent. The current prototype only reasons about the type of an agent to determine if it is suitable or not. In situations where there are more agents of the same type, or where certain actions or goals are incompatible, the assignment process will have to take the believability of the agents into account. For example, it is unbelievable for an agent to be both unconscious and conscious simultaneously. Westra et al. (2009) suggest the use of a negotiation step between the agents and the system for task assignments during the scenario [47]. Although their work focuses on centralised control over difficulty adaptation at run time, the same principle can be applied to task assignment during the construction of the scenario. When the Scenario Generator needs to assign a goal to an agent, it asks the available agents to consider, given the goals they have already been assigned, how believable it would be for them to also achieve the new goal. The system then selects the agent that is most likely to achieve the goal. If the goal cannot be assigned to any of the agents present without compromising their believability, the Scenario Generator adds a new agent to the game world.

Once a fully functioning prototype has been developed, it would be most interesting to see how much work is required to configure this prototype for various training domains and how well the prototype will perform across such different domains.

7.2.2 Task and Competency Suitability

The evaluation experiment showed that half of the scores given to scenarios generated by the system indicated them to be at least ‘tolerably suitable’ to train the task that was specified as the training objective. Regarding the competency level of the trainee, however, only the top 25 percent of the scores was at least ‘tolerably suitable’. I expect that this is the result of employing a static scenario template and a static difficulty level. Because the planner decomposes all high-level tasks specified in the scenario template, the trainee has to perform not only the target tasks, but also other tasks. This was an intentional design choice, since it is desirable to create complete scenarios where the trainee has to follow all steps of the procedure. However, in several cases this made the scenario more difficult than desired, and in some cases it was even expected that it would distract the trainee so much that he could no longer focus on the training objective. In situations where the trainee has to really focus on the target task, it might therefore be desirable to *not* create the necessity for the trainee to actually perform any other actions while still following procedure. For example, if the scenario focuses on treating a burn wound, procedures specify that the trainee will first have to secure the environment. However, if the trainee really needs to focus on the target tasks, it might be desirable to not introduce any threats in the environment. The trainee will still be required to check for threats but he will not have to react to them to accomplish the ‘secure environment’ task.

One solution to this problem would be to specify several scenario templates and annotate them with difficulty levels. A more elegant solution is proposed in the work of Zook et al. (2012) [50] as discussed earlier. Although their framework reasons about scenario generation in a completely different way, the difficulty model they employ might be transferable to my solution. The authors expect a target performance curve as input to their scenario generator. Such a curve specifies the desired trajectory of the trainee’s performance during the scenario. Based on a skill model of the trainee and complexity levels of the events, the performance of the trainee can be predicted. The aim of the system is to introduce events in such a way that the predicted performance curve follows the target performance curve as close as possible. Such a curve could also be used to steer the decomposition schema selection process of my framework. When performance has to be high, less complex decompositions are chosen, and when the trainee has to be challenged, more complex schemas are selected. This approach will allow more control over the overall flow of the scenario.

A second (complementary) extension that would allow a better fit of the events to the trainee’s competencies, was suggested by one of the evaluators during the experiment. He pointed out that in First Aid role play sessions, the trainee is seldom alone, but instead has a colleague trainee with him. Scenarios that are too complex for one trainee might be perfect for two trainees. In the system the role of the colleague could be played by a virtual agent. When necessary, the system can then assign the execution of specific tasks to this colleague agent, relieving the trainee. Which tasks have to be assigned to whom could be decided using either the difficulty levels as proposed in the system design, or the performance curve discussed above.

7.2.3 Variety

Variety between scenarios aimed at training the same task is important for effective training. Although the prototype’s capability of generating varied scenarios has not been explicitly tested, the whole framework has been designed to support variety. The upper-ontology for the domain-specific knowledge allows for quick and easy additions of new information to the knowledge base. The system can directly use any newly specified objects in the scenario. Because the scenario generation process is influenced by the objects it has selected in earlier steps, the addition of an object does not only have an effect upon the game world content, but also on the actions planned for the trainee, and the actions suggested for the virtual agents. Furthermore, other elements also increase variety allowing the generator to diversify even more. For example, it can select a different decomposition schema that imposes different requirements on the game world, which in turn influence the behaviour of the agents and the selection of additional objects. Because the size of the knowledge base directly influences the variety between scenarios, future research efforts should be directed at the development of specific tools that allow domain experts to easily extend the knowledge base without the use of system experts. The upper-ontology is a first step towards such a tool, but additional support should be given to ensure consistency, and to provide graphical overviews.

7.2.4 Comparison

Section 7.1 already discussed some other approaches that have been suggested for automated scenario generation, and the differences between those systems and the one proposed in this work. However, a true comparison of these different approaches regarding their performance in terms of quality, variety, and the configurability of the system to other training domains, is difficult. Every system requires its own problem specification and has its own result format. Moreover, the systems consider different quality requirements. Such a comparison, however, could be informative. Some approaches might turn out to better suitable for some training domains than other. Since the framework I propose is based on planning trainee actions following the domain specific procedures, it can be hypothesised that the prototype will function well for highly organised domains, but might have problems with domains where there are few rules restricting the actions of the trainee. In the latter domains the approach of Zook et al. ([50]) as discussed earlier might give better results. By combining different techniques, it might be possible to create a system that will perform well for all training domains.

Chapter 8

Conclusion

In this research project, I set out to answer the question: “Can a system automatically generate effective training scenarios for Adaptive Educational Games by coupling automated planning techniques with the use of Smart Objects?” The first step in resolving this question has been an analysis of the field of SBT. Literature studies and interviews with experienced instructors identified several requirements of an effective training scenario. Based on these requirements, a domain-configurable framework for the automatic generation of training scenarios was designed. This framework employs a hybrid HTN planner that determines the actions of the trainee and the events required to prompt these actions. The system then considers the objects that should be present in the game world to facilitate these actions and events. To allow the system to reason about game world objects, and the interaction possibilities they add to the world, the framework makes use of Smart Objects. Combining the planning of trainee actions with the selection of Smart Objects ensures that the requirements of effective training scenarios are met. Moreover, this coupling results in a unique scenario generation system that, unlike other systems, facilitates both runtime guidance and assessment of the trainee by the encompassing AEG, and automatic content generation. In order to test the framework, a prototype for the domain of First Aid was implemented, and its performance evaluated. The experiment was set up to compare the quality of scenarios generated by the system to the quality of scenarios written by human experts and laymen. It was hypothesised that the scenarios written by experts would be best, followed by those generated by the system and finally the laymen scenarios. The results, however, indicated no significant differences between any of the scenario sources, although the average scores follow the hypothesised trend. It seems unlikely that there would be no significant differences between expert and layman scenarios, and several discussion points have been identified that might have influenced these results. The predetermined contextual setting of the scenario might have had a negative influence on the human authors. Also, the format of the scenario descriptions being evaluated did not consider the course of events during the scenario. As a result the evaluators might have had too little information to truly judge the qualtiy of the scenario. Finally, the experiment underlined the importance of developing the knowledge base of the system in cooperation with a domain expert. Further experiments will be required to come up with definitive answers. For now, all that can be concluded is that the proposed framework seems to be capable of generating effective training scenarios, but further testing is required to identify its strengths and weaknesses.

Based on the results of the experiment and the analysis of newly proposed techniques for scenario generation, several additional directions for further research have been suggested.

First of all, the implementation of a fully functional prototype should facilitate the extendibility of the prototype to other trainings domains. It would be interesting to see how much work such a configuration actually requires and how well the system performs across various domains. However, it was discovered during implementation that an additional component is required to distribute planned actions over the available agents, without endangering their character’s believability. A bidding system such as proposed by Westra et al. ([47]) might provide a solution.

The experiment showed that the generated training scenarios were not always well adapted to the competency level of the trainee. A second direction for future research would therefore be the extension of the system to work with a performance curve as suggested by Zook et al. ([50]), and to allow virtual agents to take over part of the responsibilities of the trainee.

A third point for consideration is the capability of the system to generate varied training scenarios. Although this capability is assumed to be inherent to the design of the framework, it has not yet been explicitly tested. Moreover, specialised tools will have to be developed to ease the creation of extensive knowledge bases that will facilitate this variation.

Another interesting line of research would be the comparison of the different approaches suggested for automated scenario generation, regarding their scenario quality, variety, and domain configurability. Although most authors perform some kind of evaluation experiment, it is difficult to compare their approaches, because they differ regarding the scenario quality requirements, training domain, and in- and output. An objective measure should be defined along with an encompassing framework to compare the systems.

Finally, the Scenario Generator was designed to work within the context of an AEG. Only when it is fully integrated into such a game along with the necessary procedural content generation techniques, can its true potential be evaluated.

As the use of serious games in education becomes more popular, the need for effective training scenarios increases. Although further research is still required to ensure proper quality of the scenarios, the framework I present brings us a step closer to an automated scenario generation system. With the inclusion of such a system in a serious game, it will become feasible to present trainees with varied training opportunities that are tailored to their individual needs and abilities. In the future, this might offer trainees additional autonomous training opportunities that are nevertheless guided by didactic principles.

Bibliography

[1] Tolga Abaci and Daniel Thalmann. Planning with Smart Objects. In *The 13th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision: WSCG*, pages 25–28, 2005.

[2] Ruth Aylett, Rui Figueiredo, Sandy Louchart, João Dias, and Ana Paiva. Making it up as you go along improvising stories for pedagogical purposes. In Jonathan Gratch, Michael Young, Ruth Aylett, Daniel Ballin, and Patrick Olivier, editors, *Intelligent Virtual Agents*, volume 4133 of *Lecture Notes in Computer Science*, pages 304–315. Springer Berlin/Heidelberg, 2006.

[3] Karel van den Bosch and Johan B.J. Riemersma. Reflections on scenario-based training in tactical command. In Samuel G. Schiflett, Linda R. Elliott, Eduardo Salas, and Michael D. Coovert, editors, *Scaled Worlds: Development, Validation and Applications.*, chapter 1, pages 1–21. Ashgate, Aldershot, England, 2004.

[4] Clint Bowers, Florian Jentsch, David Baker, Carolyn Prince, and Eduardo Salas. Rapidly Reconfigurable Event-Set Based Line Operational Evaluation Scenarios. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 41(2):912–915, October 1997.

[5] Donald J. Campbell. Task Complexity: A Review and Analysis. *The Academy of Management Review*, 13(1):40–52, January 1988.

[6] Janis A. Cannon-Bowers, John J. Burns, Eduardo Salas, and John S. Pruitt. Advanced technology in scenario-based training. pages 365–374, 1998.

[7] Luis Castillo, Juan Fdez-Olivares, Óscar García-Pérez, and Francisco Palao. Efficiently handling temporal knowledge in an HTN planner. In *Proceedings of the 16th International Conference on Automated Planning and Scheduling: ICAPS 2006*, pages 63–72, 2006.

[8] Mihaly. Csikszentmihalyi. *Flow : the psychology of optimal experience.* Harper Perennial, 1991.

[9] Ken Currie and Austin Tate. O-Plan: The open planning architecture. *Artificial Intelligence*, 52(1):49–86, November 1991.

[10] Robb Dunne, Sae Schatz, Stephen M. Fiore, Glenn Martin, and Denise Nicholson. Scenario-Based Training: Scenario Complexity. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 54(27):2238–2242, September 2010.

[11] Kutluhan Erol, James Hendler, and Dana S. Nau. UMCP: A Sound and Complete Procedure for Hierarchical Task-Network Planning. In *Proceedings of the 2nd International Conference on Artificial Intelligence Planning Systems: AIPS 94*, pages 249–254, 1994.

[12] Jennifer Fowlkes, Daniel J. Dwyer, Randall L. Oser, and Eduardo Salas. Event-Based Approach to Training (EBAT). *The International Journal of Aviation Psychology*, 8(3):209–221, July 1998.

[13] Alfonso Gerevini, Ugur Kuter, Dana Nau, Alessandro Saetti, and Nathaniel Waisbrot. Combining Domain-Independent Planning and HTN Planning: The Duet Planner. In *Proceedings of the 18th European Conference on Artificial Intelligence: ECAI 2008*, pages 573–577, Amsterdam, The Netherlands, The Netherlands, 2008. IOS Press.

[14] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning: Theory & Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[15] Michael Steven Gold and Peter M. Bentler. Treatments of missing data: A monte carlo comparison of rbhdi, iterative stochastic regression imputation, and expectation-maximization. *Structural Equation Modeling: A Multidisciplinary Journal*, 7(3):319–355, 2000.

[16] J.E. Greer, G. McCalla, and North Atlantic Treaty Organization. Scientific Affairs Division. *Student Modelling: The Key to Individualized Knowledge-Based Instruction*. NATO ASI series: Computer and systems sciences. Springer-Verlag, 1994.

[17] Eugene Grois, William H. Hsu, Mikhail Voloshin, and David C. Wilkins. Bayesian Network Models for Generation of Crisis Management Training Scenarios. In *Proceedings of IAAI-98*, 1998.

[18] Robert J. Hall. Explanation-based Scenario Generation for Reactive System Models. In *Automated Software Engineering*, 1998.

[19] Kenneth Hullett and Michael Mateas. Scenario generation for emergency rescue training games. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 99–106, New York, NY, USA, 2009. ACM.

[20] Jesús Ibáñez Martínez and Carlos Delgado Mata. A Basic Semantic Common Level for Virtual Environments. *The International Journal of Virtual Reallity*, 5(3):25–32, September 2006.

[21] Marcelo Kallmann and Daniel Thalmann. Modeling Objects for Interaction Tasks. In *Proceedings of the Eurographics Workshop on Animation and Simulation*, pages 73–86, 1998.

[22] Subbarao Kambhampati, Amol Mali, and Biplav Srivastava. Hybrid Planning for Partially Hierarchical Domains. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 882–888, 1998.

[23] Judy Kay. Lifelong learner modeling for lifelong personalized pervasive learning. *IEEE Transactions on Learning Technologies*, 1(4):215–228, 2008.

[24] Paul A. Kirschner, John Sweller, and Richard E. Clark. Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching. *Educational Psychologist*, 41(2):75–86, June 2006.

[25] J. Richard Landis and Gary G. Koch. The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159.

[26] Ricardo Lopes and Rafael Bidarra. A semantic generation framework for enabling adaptive game worlds. In *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, ACE '11, New York, NY, USA, 2011. ACM.

[27] Ricardo Lopes and Rafael Bidarra. Adaptivity Challenges in Games and Simulations: A Survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(2):85–99, June 2011.

[28] Glenn Martin, Sae Schatz, Clint Bowers, Charles E. Hughes, Jennifer Fowlkes, and Denise Nicholson. Automatic Scenario Generation through Procedural Modeling for Scenario-Based Training. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 53(26):1949–1953, October 2009.

[29] Glenn A. Martin and Charles E. Hughes. A scenario generation framework for automating instructional support in scenario-based training. In *Proceedings of the 2010 Spring Simulation Multiconference*, SpringSim 2010, pages 35:1–35:6, San Diego, CA, USA, 2010. Society for Computer Simulation International.

[30] T. L. McCluskey, D. Liu, and R. M. Simpson. GIPO II: HTN Planning in a Tool-supported Knowledge Engineering Environment. In *Proceedings of the thirteenth International Conference on Automated Planning and Scheduling*, pages 92–101, 2003.

[31] Eva Millán, Tomasz Loboda, and Jose L. Pérez-de-la Cruz. Bayesian networks for student model engineering. *Computers & Education*, 55(4):1663–1683, December 2010.

[32] Dana S. Nau. Current trends in automated planning. *AI magazine*, 28(4):43–58, 2007.

[33] Dana S. Nau, Tsz-Chiu Au, Okhtay Ilghami, Ugur Kuter, J. William Murdock, Dan Wu, and Fusun Yaman. SHOP2: An HTN Planning System. *Journal of Artificial Intelligence Research*, 20:379–404, dec 2003.

[34] James Niehaus and Mark O. Riedl. Scenario Adaptation: An Approach to Customizing Computer-Based Training Games and Simulations. In Scotty D. Craig and Darina Dicheva, editors, *Proceedings of the AIED 2009 Workshop on Intelligent Educational Games*, volume 3, pages 89–98, 2009.

[35] Geoff Norman. Likert scales, levels of measurement and the laws of statistics. *Advances in Health Sciences Education*, 15:625–632, 2010. 10.1007/s10459-010-9222-y.

[36] Randall L. Oser, Janis A. Cannon-Bowers, Eduardo Salas, and Daniel J. Dwyer. *Enhancing Human Performance in Technology-Rich Environments: Guidelines for Scenario-Based Training*, volume 9 of *Human/Technology Interaction in Complex Systems*, pages 175–202. Jai Press Inc., 1999.

[37] M. Peeters, K. Bosch, J. J. Meyer, and M. Neerincx. Situated cognitive engineering: the requirements and design of directed scenario-based training. In *5th International Conference on ACHI*, pages 266–272, 2012.

[38] N. Peirce, O. Conlan, and V. Wade. Adaptive Educational Games: Providing Non-invasive Personalised Learning Experiences. In *Digital Games and Intelligent Toys Based Education, 2008 Second IEEE International Conference on*, pages 28–35. IEEE, November 2008.

[39] Mark O. Riedl and R. Michael Young. Narrative Planning: Balancing Plot and Character. *Journal of Artificial Intelligence Research*, 39(1):217–268, September 2010.

[40] Eduardo Salas, Heather A. Priest, Katherine A. Wilson, and C. Shawn Burke. *Senario-Based Training: Improving Military Mission Performance and Adaptability*, volume 2 of *Military Life: the psychology of serving in peace and combat*, chapter 3, pages 32–53. Praeger Security International, 2006.

[41] Patrick E. Shrout and Joseph L. Fleiss. Intraclass Correlations: Uses Assessing Rater Reliability. *Psychological Bulletin*, 86(2):420–428, 1979.

[42] R. M. Smelik, T. Tutenel, K. J. de Kraker, and R. Bidarra. A Declarative Approach to Procedural Modeling of Virtual Worlds. *Computers & Graphics*, 35(2):352–363, April 2011.

[43] J. M. Thomas and R. M. Young. Annie: Automated Generation of Adaptive Learner Guidance for Fun Serious Games. *Learning Technologies, IEEE Transactions on*, 3(4):329–343, October 2010.

[44] Tim Tutenel, Rafael Bidarra, Ruben M. Smelik, and Klaas J. De Kraker. The Role of Semantics in Games and Simulations. *Comput. Entertain.*, 6(4):1–35, December 2008.

[45] Jurriaan Van Diggelen, Tijmen Muller, and Karel Van Den Bosch. Using artificial team members for team training in virtual environments. In *Proceedings of the 10th International Conference on Intelligent Virtual Agents*, IVA'10, pages 28–34, Berlin, Heidelberg, 2010. Springer-Verlag.

[46] L. S. Vygotsky. *Mind in Society: The Development of Higher Psychological Processes*. Harvard University Press, 1978.

[47] Joost Westra, Hado van Hasselt, Frank Dignum, and Virginia Dignum. Adaptive serious games using agent organizations. In Frank Dignum, Jeff Bradshaw, Barry Silverman, and Willem van Doesburg, editors, *Agents for Games and Simulations*, volume 5920 of *Lecture Notes in Computer Science*, pages 206–220. Springer Berlin/Heidelberg, 2009.

[48] Robert E. Wood. Task complexity: Definition of the construct. *Organizational Behavior and Human Decision Processes*, 37(1):60–82, February 1986.

[49] R. Michael Young, R. Michael Young, Martha E. Pollack, Martha E. Pollack, Johanna D. Moore, and Johanna D. Moore. Decomposition and Causality in Partial-Order Planning. In *Proceedings of the 2nd International Conference on AI Planning Systems: AIPS-94*, pages 188–193, 1994.

[50] Alexander Zook, Stephen L. Urban, Mark O. Riedl, Heather K. Holden, Robert A. Sottilare, and Keith W. Brawner. Automated scenario generation: toward tailored and optimized military training in virtual environments. In *Proceedings of the International Conference on the Foundations of Digital Games*, FDG '12, pages 164–171, New York, NY, USA, 2012. ACM.