Using scenario smells to analyze scripted communication scenarios in virtual learning environments

Timo Overbeek

Supervised by:
Johan Jeuring
Raja Lala

May 2019

Contents

1 Introduction **4**

2 Communication Education **7**

2.1 Communication Training Software 7

2.2 Dialog Scenario Language 9

2.3 Communicate 10

3 Hypothesis and research question **15**

3.1 Research Question 15

3.2 Scenario Smells 17

3.2.1 Example 1 17

3.2.2 Example 2 18

3.2.3 Scenario Smell 19

3.3 Displaying Information 21

3.4 Research Method 22

4 Interviews **24**

4.1 Interview Creation 24

4.2 Interviewee 26

4.3 Results 27

4.3.1 Current Use 27

4.3.2 Valid Scenarios 28

4.3.3 Optimal Path 31

4.3.4 Proposed Improvements 34

4.4 Conclusion 39

4.5 Scenario Smells 40

5 Implementation **42**

5.1 Calculating the Optimal Path 43

5.2 Size of Search Space 44

5.2.1 Basic Combinators 44

5.2.2 Interleave Points 48

5.3 Subject Monotonicity 52

5.4 Calculating Scenario Smells 55

5.5 Scenario Smells Report 58

6 End Interviews **59**

6.1 Introduction 59

6.2 Usability of Scenario Smells 60
6.3 Calculation Time 63

7 Evaluation **65**

7.1 General Scenarios 66
7.2 A scenario with sequential subjects 68
7.3 A scenario with parallel subjects 69
7.4 A scenario with high subject complexity 71

8 Discussion **73**

8.1 Answer Research Question 73
8.2 Recommendations 76

9 Future Research **79**

9.1 Workflow of Scenario Writers 79
9.2 Pruning 80
9.3 Learning Effect 81

10 Conclusions **83**

Chapter 1

Introduction

Face-to-face communication is an essential skill in many professions. Doctors need to convey bad news to their patients; teachers inform parents about the progress of their pupils; managers hold salary negotiations with their employees; prison guards need to assess the aggression levels of detainees correctly.

Many educational programs include communication skills in their curriculum objectives. There are many ways these learning objectives can be reached, but practicing these skills with role-playing sessions seems to be a particularly effective method [1]. In these sessions, two or more students perform a conversation. They each take on a predefined role, which can either be that of a working professional or that of a client. A third person often watches the conversation and provides feedback once the conversation is done.

To practice communication skills, a student can use a serious game. Such a game often simulates role-playing sessions, but only one role is performed by a player and the others are performed by the computer. Some communication training software allows a player to write his statements [10], but most games use scripted conversations [2, 3, 6]. In a scripted conversation, a player selects an answer from a limited set of provided answers.

Communication experts possess the knowledge necessary to develop a scripted conversation. To make developing scripted conversations as easy as possible,

a communication expert can use a special-purpose editor for developing scenarios. A game can then include the conversations developed by a communication expert. Using a special-purpose editor for developing conversations means that authors do not need programming skills.

Directed acyclic graphs (DAGs) are a popular way to structure a dialog in communication software [7]. In such a DAG every node represents a statement, and the edges indicate the flow of the conversation. There are many possible paths that can be taken through a DAG, and every path needs to result in a plausible conversation. Depending on the size and complexity of the dialog, this can make writing the required DAGs a complicated process.

Some games provide feedback to a player in the form of a score [**?**, 6]. This may make writing a DAG even harder because an author will not only need to make sure that a dialog is always plausible, but also that a scoring correctly represents the player's performance for every player choice. For example, if every NPC node has one preferred player follow-up, and those preferred player statements all have the same score, then the longest path will always result in the highest score. The length of a path does not necessarily tell us anything about its desirability and is therefore probably not a correct way to measure player performance. A long path that yields a high score might be desirable, but could also be an indication that something is wrong with the scenario.

In computer programming, we sometimes talk about code smells [4]. With a code smell, we refer to a common symptom of a certain error in a piece of code. This error might be on a much deeper level than the symptom. For example, the occurrence of very large classes is often referred to as a code smell. There might be situations where a large class is required, but they are often a symptom of bad class hierarchy. The duplicate code A code smell does not necessarily imply bad code. However, a code smell is often much easier to find than the actual problem, and code smells can, therefore, help find these problems. As discussed before, if the longest path in a scenario yields the highest possible score, there might be a problem. The scoring could still be correct, the longest path might also be the most desirable path, but this occurrence often indicates that something in the scenario is wrong. This principle is similar to a code smell, and we will, therefore, refer to this as a scenario smell.

We hypothesize that identifying scenario smells helps a scenario author to create better training scenarios. In this study, we will attempt to identify different kinds of scenario smells. We will also explore different ways in which we can notify a scenario author of scenario smells. We will look both at the desirability of these solutions and their technical feasibility.

We organized this study as follows. In chapter 2 we discuss the current state of communication education by looking at the literature that is available on the subject. In chapter 3 we explain our hypothesis of the perceived problems and their possible solutions in further detail. In chapter 4 we describe various interviews we conducted to verify our hypothesis. In chapter 5 we discuss the technical possibilities and limitations of our solutions. In chapter 6 we describe a second set of interviews we conducted. In chapter 7 we discuss the practical performance of our methods. In chapter 8 we will make some recommendations for Communicate. In chapter 9 we will discuss possible further research. We will conclude our research in chapter 10.

Chapter 2

Communication Education

2.1 Communication Training Software

In this section we give a brief overview of different serious games that support practicing communication skills. We focus on serious games that simulate a conversation. In particular, we discuss the way a dialog is structured and how decisions are made for an NPC.

Cláudio et al. [3] describe an application that teaches Pharmaceutical Science student how they should communicate with their patients. The application simulates a conversation between a player and one NPC. The game uses scripted conversations and scenarios can be created in the Dialog Creator Interface. The dialog is represented with a DAG, where every node represents one statement. Statements alternate between player and NPC, and every NPC node connects to exactly three player nodes. Player nodes have a score parameter, which is used to generate a total score at the end of the game.

Bosse et al. [2] propose a way to make NPCs more realistic and less predictable by using a cognitive model and an aggression parameter. They start from an application much like that of Cláudio et al. [3], where a dialog is represented with a DAG. Like Cláudio et al. they alternate between player and NPC nodes, and every NPC node is connected to a fixed amount of player

nodes. They offer four connections, which are matched with the following intentions: letting go, supportive, directive, and call for support. Bosse et al. improve on this initial game, by adding an aggression parameter. The value of this parameter is influenced both by prior statement choices and the player's real-life emotions. Every player node can be connected to multiple NPC nodes and the computer makes a choice between them by looking at the aggression parameter.

Wauters et al. [10] also make use of emotions in their deLearyous application. They use the interpersonal circumplex model [9], which models behavior on two axes; one axis representing submissive versus dominant, and one axis representing cooperative versus antagonistic behavior. Submissive and dominant behavior result in the opposite behavior in a conversation partner. Cooperative and antagonistic behavior result in similar behavior in a conversation partner. DeLearyous leaves the player free to create his/her own statements, and uses a natural language processing (NLP) module to process a player statement. The NLP module classifies where on the circumplex model a statement belongs, and also identifies the keywords in the player's statement. A finite state machine then combines these two to select a NPC statement.

Communicate [6] separates its scenarios into subjects. Each subject is in turn represented by one DAG. Subjects need to be traversed in a predefined sequence. However, subjects can be interleaved, in which case the player is free to choose between statements of the interleaved subjects. This process is illustrated in figure 2.1. The vertical axis represents the sequence of subjects. Subjects on the same horizontal line are interleaved, and can be traversed in any order. All the subjects on one horizontal line need to traversed, before the subjects on the next line become available.

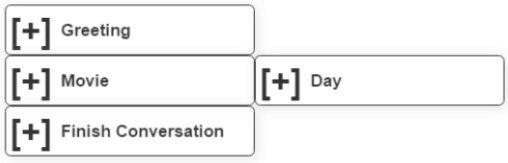

Figure 2.1: An example of a subject hierarchy within Communicate

Communicate uses emotions in the animations of the NPCs, but not for determining the flow of the conversation. The dialog flow can be controlled

by means of score parameters, which can be defined by the scenario author. A player node can change score parameters. A player node can also have one or more preconditions. A precondition allows a statement to be presented only if the precondition is satisfied, for example if the scoring parameters are within a certain range.

Leuski and Traum [8] propose to use a virtual human for other purposes than communication training, such as guiding a tour, or perform secretarial services. Like Wauters et al. [10] they use an NLP module for dialogs. They do not use any control flow for dialogs. Instead the statements of the NPC will try to give a correct answer to a player's statement, by using an individual knowledge base. Different NPCs might have different knowledge bases, which results in different behavior.

Gebhard et al. [5] describe Visual Scenemaker, a tool that can be used to create interactive applications with multiple NPCs. Like the application of Leuski and Traum, Visual Scenemaker is not specifically build for communication education, but it can be used for that purpose. Gebhard et al. separate the content, dialog statements, from the logic that describes the dialog flow. They use a directed graph to create a dialog flow, but unlike the other examples the graph does not have to be acyclic.

2.2 Dialog Scenario Language

There are a lot of similarities between the discussed applications. All the applications that use a scripted dialogue make use of a graph to represent the dialog flow. In most cases this graph takes the form of a DAG. Many applications also make use of parameters that can be increased or decreased during the conversation. Scenario author can use these parameters as a feedback opportunity or as a way to represent emotions. In quite a few applications the parameters also allow the scenario author to exercise more control over the dialog flow.

Lala et al. [7] compare multiple applications and define a language that can be used to compare different scripted conversation simulations. They define the entire script as a scenario, that can consist of multiple subjects.

Every subject in turn consists of a DAG. Nodes in the DAG can have multiple properties, the most important one is the statement, which contains the actual dialog. Other components are also possible, for instance parameter increases, emotion changes, player feedback or preconditions. The authors also define an XML scheme called Utrecht University Dialogue Scenario Language (UUDSL) that can be used to describe the structure of a scenario.

In this study we use UUDSL as much as possible. This makes our research applicable in a wide array of communication training software. We use the Communicate software as benchmark, because we have full access to its code. We now provide an overview of the functions of Communicate for the unfamiliar reader.

2.3 Communicate

Communicate is a web based application that consists of a game and a scenario editor. In the game students have a conversation with a single NPC. The NPC statements are displayed in a text balloon and the possible player statements are displayed at the bottom of the screen.

The game measures the performance of a player with scores. Each score represents a skill or communication goal. For example, in a scenario where a player needs to discuss with a NPC which movie they are going to see, the game can measure the player performance in a goal score and a relation score. The goal score indicates if the player succeeded in selecting the movie he/she wanted. The relation goal shows how angry or happy the NPC is with the player.

Choosing certain player statements will increase or decrease the scores. A scenario author can decide if these score changes are immediately visible or if they remain invisible. Either way, at the end of the game a player sees the final scores, and a total score that is the weighted average of all the scores. In the same screen a player also receives some feedback and can review the entire conversation history.

Scenarios can be created in the scenario editor. When the editor is opened

Figure 2.2: An example screen of Communicate in use

an overview of the different subjects is shown. Figure 2.1 shows this screen. Subjects on the same horizontal level are interleaved, which means they can be traversed in any order. The vertical levels show the sequence of subjects. A player needs to traverse all the subjects on a horizontal level before he/she can progress to the next horizontal level.

The scenario author can click on a subject to view the DAG that represents the subject. Figure 2.3 shows this screen. Red nodes are NPC nodes and blue nodes are player nodes. The green node is the currently selected node and the details of the node are displayed on the right-hand side of the screen. Nodes that have no edges connected to them are nodes that can start the subject. Multiple starting points are possible. Nodes that have no outgoing edges end the subject, and allow a player to select a new starting node from the currently available subjects.

NPC nodes can have three special labels. Firstly, they can be labeled as the end of the conversation. This will end the game even if there are still untraversed subjects. Secondly, they can be labeled as an early end point of the subject. In that case the player can choose to continue with the current

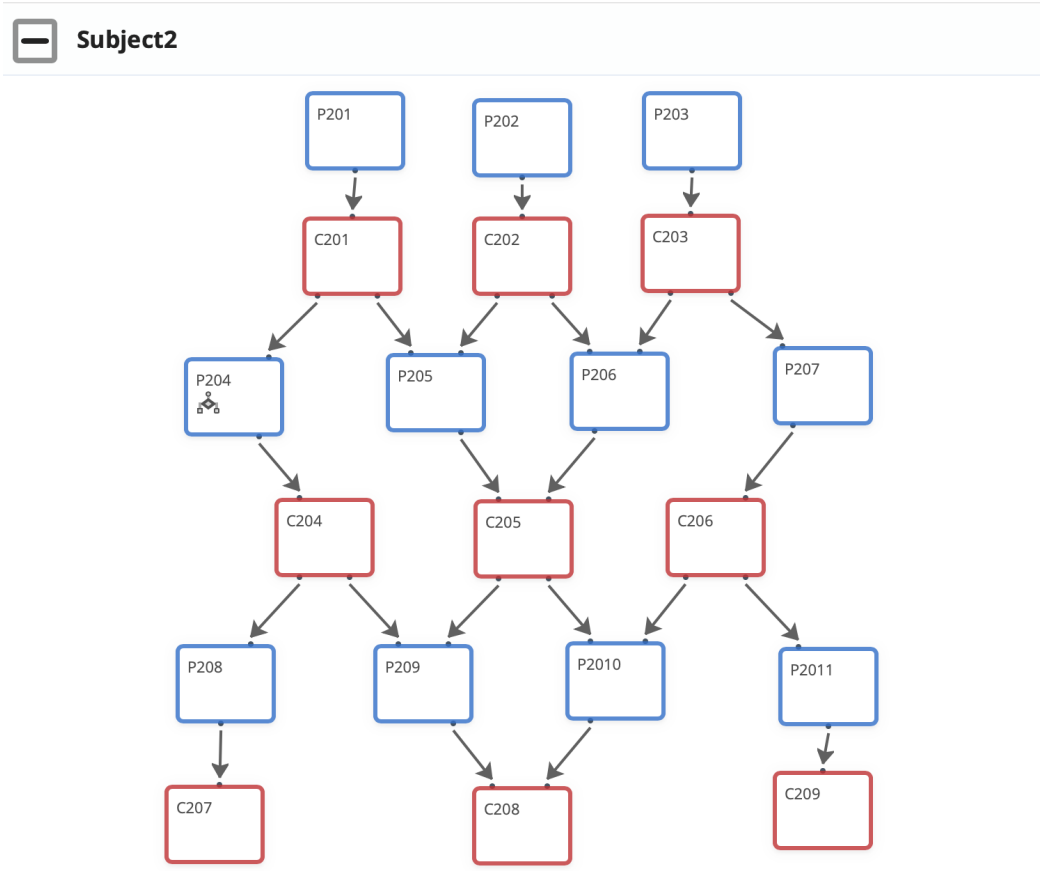

Figure 2.3: An example of a DAG for a subject in Communicate

subject by choosing one of the connected player nodes. The player can also choose one of the starting nodes of the other available subjects. Choosing the second option will end the current subject. Thirdly, a node can be labeled as a point on which the subject might be switched. This functions in a similar way as the early end to subject option, apart from the fact that the subject is not ended. Instead the connected player statements become available again the next time the player needs to choose a new subject.

A lot of functions in Communicate are controlled with parameters. The entire scenario uses the same set of parameters. The scenario author can add parameters to and remove parameters from this set. Every parameter has a name and a type. The type can be an integer, a string, a boolean

Figure 2.4: An example of a feedback screen within Communicate

or an enumerate. An Integer has a minimum value, maximum value, and start value. Integers can also have a weight, which marks them as a scoring parameter. The weighted total score is calculated using these weights.

Both scoring parameters and non-scoring parameters can be used to control the dialog flow. Nodes can have preconditions. A node will only be available if its preconditions are true. Preconditions take the following form: [parameter name] [$=$, $\neq$, $<$, $>$, $\leq$, $\geq$] [value]. If there are multiple preconditions a scenario author can select if all preconditions need to be true or if one precondition needs to true. Preconditions can be grouped together in which case they are interpreted as a single precondition (being either true or false) on the level above their own.

Both player and NPC nodes can have parameter changes attached to them. Parameters can be increased or decreased with a certain value. Parameters can also be set to a specific value. Emotions roughly work the same way. We will therefore largely ignore them, and assume that everything that is

possible with parameters can be replicated with emotions.

Chapter 3

Hypothesis and research question

3.1 Research Question

In our literature study we discussed that many serious games for communication education use DAGs to represent a scripted dialog. This is not surprising, because DAGs are easy to understand for humans and traversing a DAG does not require a lot of computer resources. However, when a DAG becomes larger, it can become difficult to maintain an overview of the entire graph.

To make maintaining an overview easier we would like to keep the visualisation of a node small. This allows us to see a larger part of the graph at the same time. If an individual node has lots of properties, we will need to make a choice between displaying all the information and keeping the node small. A common solution to this problem is to have some information always visible within the node, and have some information only visible when a node is selected. The problem with this solution is that it becomes more difficult to see how some properties are influenced by multiple nodes.

Communication education software often uses parameters that can be in-

creased or decreased by individual nodes. In that case all the traversed nodes have some some influence over the final score. It is therefore important to keep an overview over the relation between different score increases and decreases. The large amount of possible paths through a DAG can make this hard. If some properties are hidden to improve overall visibility, this becomes even harder.

The goal of this study is to make it easier to write scenarios. It is our hypothesis that assigning scores is the most difficult part of creating scenarios. It can be hard to display all the relevant information, and scenario authors have difficulty with combining smaller pieces of information in scenario wide information. We will discuss different ways in which information can be displayed in a DAG. This includes providing combined information, like the value of a parameter over an entire path. We belief that a special focus should be on scenario smells. Scenario smells are possible indications of errors in the DAG. We want to notify the user of these scenario smells, so that the scenario author can check if there is indeed is an error.

This hypothesis leads to the following research question:

- Can we provide scenario authors with the information they deem important for the improvement of scenario quality?

To answer this question, we want to answer the following sub-questions:

- Is scoring one of the more difficult parts of creating a scenario?
- What information might help with improving the scoring?
- What information can be calculated, both in theory and in practice?

In the rest of the chapter we explain our hypothesis more. We discuss why we assume that scoring a DAG is difficult and how we think we can help scenario authors. At the end of the chapter, we introduce our research method. We provide the research results in chapter 4 to 7.

3.2 Scenario Smells

It is our hypothesis that the scoring of a scenario is important for the playing experience. The scenario author creates a scenario with a certain learning goal in mind. This learning goal often dictates which player options are preferred and which are undesirable. The player has implicit expectations about what certain scores mean. It is important that the scores the player gets, correctly represent how the scenario author values the players performance. Otherwise the player can get frustrated or he/she can wrongly assume that a certain wrong approach is correct. Either way, the learning goals will not be accomplished.

There any many different paths through a scenario, and every path has its own final scores. Most nodes lie on multiple paths. These nodes will therefore contribute to multiple final scores. The scenario author needs to assign each node a score change that provides the correct final score for every possible path. This can be difficult. We illustrate potential difficulties with two examples.

3.2.1 Example 1

We display the graph of our first example in Figure 3.1. Based on this example we can assume that the scenario author favours player option B, D, and F, because those three nodes lead to immediate score increases. By that same measure we can assume that the author does not prefer node C, E, and G. Both node B and D lead to a score increase, so we might conclude that path A-B-D is preferred by the scenario author.

The path A-C-F leads to a final score of 20, which is the highest possible score. We call this path an optimal path. We expect that a preferred path is also an optimal path. In our example this does not seem to be the case. This can have two explanations. Firstly, the scenario author might have made a mistake. It could have been the intention to make A-B-D the optimal path, but the scenario author assigned too high a score increase to node F. Secondly, the assumption about the fact that A-B-D is the preferred path,

might be wrong. The author might have intended that option B provides a quick win, while option C makes the player work harder for a larger final reward.

We do not know if the first or second explanation is true. This is an example of a 'scenario smell'. Our assumptions seem to indicate that the scoring might be wrong. However, the scoring might be correct, if our assumptions are wrong. A scenario smell can therefore not be used to prove errors, but it can be used to find them.

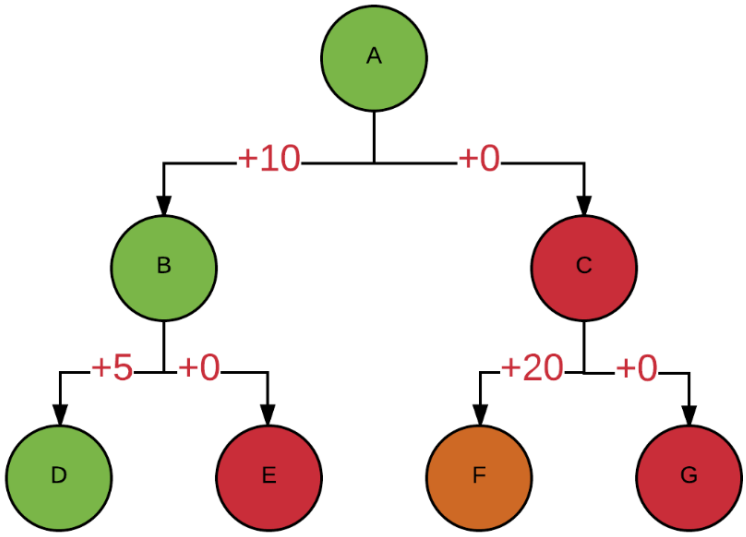

Figure 3.1: DAG of example 1

3.2.2 Example 2

Our second example uses the DAG displayed in Figure 3.2. This example shows a design pattern that is used quite often in scenarios. The player first needs to make a choice between a good option (B) and a bad option (C). If the player choose the wrong option, he/she often receives a chance to make up for their mistake. This is the case in our example. Even if the player chooses B, the wrong option, he/she still has a chance to reach the positive end node I.

The problem with the DAG in Figure 3.2 is that path A-B-I yields 10 points, while path A-C-E-G-I yields 15 points. In other words, choosing the wrong option C, is the best choice the player can make at node A. An explanation for why this error exist, is that every positive choice adds 5 points to the final score, and every negative choice does not change the score. A-C-E-G-I has a higher score because the path is longer, and the player therefore has more chances to choose a positive node.

Scenario authors indicate that they often accidentally make the longest path the optimal path. Like in example 1, however, we cannot state with certainty that this is the case in figure 3.2. We can think of examples where the displayed DAG would be correct. In a salary negotiation the player might need to refuse a salary offer, in order to receive a better one down the line. The only thing we can state, is that there is scenario smell if the longest path is also the optimal path. The smell might indicate an error, but is does not have to.

3.2.3 Scenario Smell

Our examples illustrate that it is easy to make mistakes in assigning scores to a DAG. The examples also illustrate that these mistakes can lead to unintended effects in the game. Lastly, the examples illustrate that it is often impossible to say with certainty whether or not a mistake has been made.

We introduced three different definitions that can be used to describe scenarios. Firstly, we have the preferred path, which is the sequence of nodes that, in the scenario author's view, represents the preferred way to perform a conversation. Secondly, we discussed the optimal path, which represents the sequence of nodes that leads to the highest possible score. Thirdly, we coined the term scenario smell, which refers to a symptom of a common mistake in scenario writing, without guaranteeing that there is a mistake.

Our examples show two different kinds of scenario smells. A scenario smell is based on assumptions. In both cases we start with the assumption that the preferred path and the optimal path should be the same. In the first example we also have another assumption. We assume that if we always select the node with the highest score increase, we will have the preferred

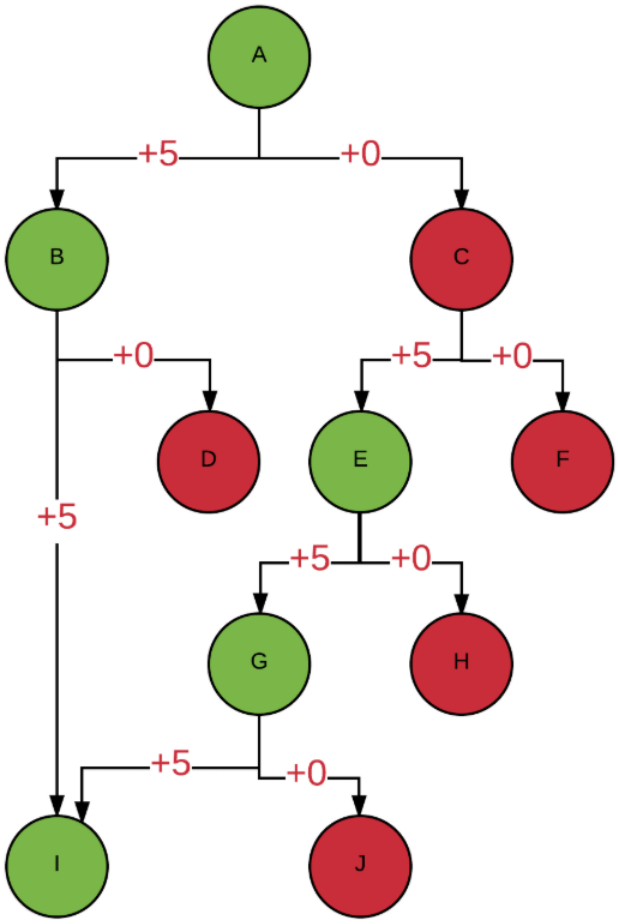

Figure 3.2: DAG of example 2

path. This is an assumption that does not always hold true. However, if the assumption holds true often enough, it can still provide us valuable feedback. In such cases a scenario author needs to make sure that the assumption is true for their own scenario, unless there is a compelling reason to ignore the assumption.

The second example shows a common mistake. A scenario author often accidentally turns the longest path into the optimal path. To avoid this problem we could state that the longest path should never be the optimal path, but that is not necessarily always true. It is better to state that if the longest path is also the optimal path, that can be a sign of bad scenario

design. In other words, it is a scenario smell.

Other kinds of scenario smells probably exist. For example, it is a common assumption that the maximal score, as defined by the scenario author, when he defines the parameter, is equal to the highest possible score. If these differ there might be something wrong with a scenario. However, this does not always have to be the case. For instance, in a medical simulation the player might need to warn the patient about possible side effects. This warning can be given at different moments in the conversation, and might even be given multiple times. Only one warning is required, however. In such a case it would make sense to cap the maximum score, to ensure that giving multiple warnings will not have a positive effect. Another possible scenario smell might be that parameters should not have a lot of correlation, because then they measure the same skill. Some scenario smells are more useful than others. In chapter 4 we discuss the different kinds of scenario smells and how we assess their usefulness.

3.3 Displaying Information

There are many ways in which creating scenarios can be made easier. We can create scenario editing tools that are easier to use, design better teaching methods to explain the editing tools, create design principles that we have proven to be effective, and let players provide feedback on a scenario while they are gaming. We focus on providing more information to a scenario author. The question is what information we need to provide, and how we visualize that information.

If we visualise a lot of information there is a risk of information overload. We therefore not only need to decide what information to show, but also when to show it. We categorise the timing of information in three ways. Firstly, some information is always visible. For instance, in the current Communicate editor, the edges connecting nodes are always visible. Secondly, there is information that can be turned on and off. For example, there might be a function that, when turned on, displays the parameter changes for every node. Thirdly, there is information that is only available when specifically asked for. In this last case requesting new information results in the removal

of the old information. For instance, in the Communicate editor, there is a function that can show the editor the parents of a selected node. Only one node can be selected at the same time, and when the editor requests the parents of another node, the old information will disappear.

A simple way to provide more information, is to make more node properties always visible. For instance, we can display the score changes inside the nodes of the DAG. This creates the risk of the earlier discussed information overload. The information overload can be decreased by collecting all the separate score changes in one total score change, or by letting the author turn the visibility of score changes on and off.

Displaying more information does not necessarily make it easier to assess how the different nodes influence each other. We think that a scenario author not only needs information about individual nodes, but also about the different paths through the tree. There are an exponential amount of paths, however, so displaying this information might be difficult.

To limit the amount of path information, we might, for instance, only display the optimal paths. This function might be expanded upon, by letting the author select a node, from which the optimal should be calculated. We can show the least optimal paths instead. Both cases might help scenario authors, because they can check how well the optimal paths and least optimal paths correspond with their preferred path and least preferred path.

Lastly, it is possible to provide statistics about subjects or the entire scenario. The three most useful statistics are probably the minimal value of a parameter, the maximal value of a parameter, and the correlation between different parameters. There are other statistics that might be useful, like the deviation in possible scores over all the paths.

3.4 Research Method

In the previous sections we discussed our hypothesis that small errors in the scoring parameters of a scenario can lead to unexpected and undesired effects. We also hypothesized that finding these errors is difficult, but can be

made easier if a scenario editing tool automatically detects so called scenario smells. It is important to verify that this hypothesis is true. We need to verify that scoring is indeed a problem, maybe even the biggest problem, with creating a scenario. We also need to verify that our proposed solutions are seen as helpful by scenario authors.

We verify our hypothesis by holding multiple interviews with current users of a scenario editor. We will ask how they create scenarios, what kind of problems they experience with creating scenarios, and what they think of our proposed improvements. In the same interview, we will also try to get a better picture of the quality of a scenario. Chapter 4 describes these interviews.

Our goal is to propose improvements for the current authoring tools. An improvement needs to be useful and implementable. The focus of our study will be on finding out what information can be calculated about a DAG. We will look both at the theory of the problem, and at practical implementations. These calculations will help us inform what kind of improvements can be made. Chapter 5 discusses the various calculations we want to develop.

The next step is to combine the information about desirability with the information about technical feasibility. We give several recommendations to designers of scenario authoring tools. These recommendations can be implemented in multiple different authoring tools. They should also be a good guideline for creating new authoring tools. We apply our own recommendations to Communicate, to demonstrate how they can be used. Chapter 8 gives these recommendations.

Lastly, we test if our recommendations satisfy the research question. We do this by returning to our original interviewee. We will demonstrate a prototype and ask them a few question about our recommendations. We will ask them if they intend to use our improvements and how their scenario design will be influenced by said improvements. We realise that an argument can be made that the improvements can only be tested with a pre- and posttest. However, this would mean that our study will only be applicable for a specific scenario authoring tool, and with very specific improvements. This motivated our choice for holding closing interviews instead. In Chapter 6 we explain the closing interviews in more detail and discuss their results. In section 9 we make some recommendations for future research.

Chapter 4

Interviews

In the previous sections we discussed our hypotheses about the kind of problems that scenario authors experience in creating good scenarios. Before we discuss possible solutions, it is important to find out if this hypotheses hold in the real world. To determine this we decide to interview 6 scenario authors with experience in writing scenarios for Communicate. We first discuss how we decide on the type of questions in the interview. Then, in Section 4.2, we discuss the results of the interviews. Lastly, in Section 4.3, we will summarize the conclusions from performing the interviews. The exact questions of the interview can be found in Appendix I.

4.1 Interview Creation

An interview consists of a mix of open and closed questions. The closed questions allow us to combine different interviews, by collecting the various answers in a single score. The risk of closed questions, is that they might steer the thought process of an interviewee in a certain direction. For instance, we want to know if the interviewee identify certain problems with scenario creation, before we have mentioned them. We therefore include some open questions at the start of the interview.

There are a couple of questions we want to answer with these interviews:

- Is scoring a substantial problem in the creation of scenarios?
- What kind of assumptions do the interviewees have about correct scenarios? If enough people have the same assumptions, would it be helpful if we notify scenario authors when those assumptions are not true for their own scenarios?
- Assuming that the scenario correctly represents the intentions of the scenario author, and therefore that the preferred path is also an optimal path, what definition of the optimal path do we need to use to calculate the preferred path?
- What kind of information would help a scenario author in creating better scenarios?
- How does a scenario author want to receive that information?

The interview is split in 6 different parts. In the first part we collect some general information about the interviewee. This mostly relates to how they use Communicate. In the second part we ask a few open questions about problems they experience while creating scenarios. Until this point, the interviewee has not been notified of our specific research question. The third part consists of closed questions about their assumptions of what makes a scenario correct. In the fourth part we ask them a couple of closed questions about what they consider to be the optimal path. The underlying question here is whether or not there is a mathematical definition for the optimal path, where the optimal path matches with the preferred path in correct scenarios. In the fifth part we propose multiple enhancements to Communicate, and ask our interviewee to review them. Lastly, in the six part, we give our interviewee the chance to speak freely about what they think should be changed in Communicate.

4.2 Interviewee

We briefly discuss the interviewees themselves. To preserve their anonymity we will name them interviewee 1 to 6. All of the interviewees have some experience with Communicate, but their experience differs from only creating a handful of simple scenarios to running a company specializing in creating such scenarios. Most of them have no prior experience with other communication training software, but do have some experience with real-life roleplaying sessions

We interviewed three university teachers (interviewee 1, 3 and 6). All of them teach a course on communication skills at Utrecht University. The educational programs to which they contribute are medicine, pharmacy and veterinary medicine. They all created a few scenarios and used them in their courses. The complexity of the scenarios differ quite a lot. Interviewee 1 is known for creating complex and extensive scenarios. The developers of Communicate often test new features on one of his scenarios, to make sure that a new feature can handle the more complex scenarios. Interviewee 3 is the opposite in that her scenarios are a lot simpler and do not make use of subjects or preconditions. Interviewee 6 holds the middle ground and uses subjects, but not preconditions.

Interviewee 2 is a student who uses Communicate as part of a honors program. She created a few scenarios about general subjects, like a bad news conversation and giving someone feedback. She herself did not use any of the scenarios in a course, but interviewee 6 developed them further, and did use some of them in a course. Interviewee 2 also had no prior experience in creating roleplay scenarios, so we conclude that she probably has the least experience in creating scenarios.

Our fourth and fifth interviewee are employees of DialogueTrainer B.V., a company that specializes in creating Communicate scenarios and that is actively involved in the development of Communicate. Unlike the other interviewees, they mostly develop scenarios for professionals, such as health-care personnel and managers, who are already working in their respective field. They often work together with players in improving a scenario. They have created more scenarios than the other interviewees (more than a dozen each)

and can be considered experts.

4.3 Results

4.3.1 Current Use

At the start of the interview we asked some questions about the current use of Communicate by the interviewee. Some users of Communicate know much more about scenario development than others. For instance, when asked about their use of the view parents and view parameters functions, multiple users indicated that they were not aware of those functions. Meanwhile, others noted that the view score function was overly complicated and difficult to use. Interestingly, the people who are aware of these functions said they used them quite often. Moreover, multiple users who did not use the functions, told us they had troubles with assigning scores. Two users even went so far as printing out their scenarios and calculating the possible end scores by hand. This seems to indicate that creating scenarios could be made a lot easier by providing better instructions and an easier user interface.

We also asked the interviewee how they went about creating scenarios. They all told us that they often create the dialogue first and only add the parameters and emotions later. The interviewees differ in how much they split up these tasks. Some go so far as to write out the entire dialog in a word document before they even start working in Communicate. Others create subparts of the scenario (dialog lines or entire subjects) and then add the parameters.

Quite a few interviewees told us that they first create a perfect dialog path, and then add possible mistakes. One person told us he played his own scenarios many times to check if the dialogue feels natural, and he then makes incremental changes to the dialog to make the dialogue feel more natural. Another person uses a similar principle, but uses test subjects to assess if the flow of the dialogue feels natural.

4.3.2 Valid Scenarios

In the next part of the interview, we asked our interviewee what, in their mind, constituted a valid scenario. To structure the outcome we prepared some statements beforehand and asked them to score the statement with a number between 1 and 5, where a 1 means they completely disagree with the statement, and a 5 means they completely agree with the statement. Interviewees often had difficulties in assigning an exact score to a statement. In a few cases we therefore had to give the interviewee the possibility to give a textual answer. We then interpolated a score from their total answer, which we offered as a suggestion for their final answer. We think that these problems do not have a large impact on the final conclusions because they did not occur too often. The interviewees did not waver between two extreme answers, but only between small differences, like a score of 1 or 2. The exact scores can be found in 4.1.

Question	I	II	III	IV	V	VI	Total
12	1	2	1	1	1	1	12
13	4	5	5	4	5	4	4.5
14	3	1	-	3	3	5	3.0
15	5	4	5	5	5	5	4.8
16	5	3	3	5	4	5	4.2
17	5	4	4	5	4	5	4.5
18	2	2	5	3	1	1	2.3
19	2	1	1	1	3	1	1.5
20	2	4	5	1	1	1	2.3
21	3	5	5	1	2	1	2.7
22	4	1	-	-	2	1	2.0
23	1	5	-	1	1	5	2.6
24	1	5	1	5	2	5	3.0
25	1	3	5	2	1	5	2.9
26	2	1	1	1	1	5	1.8

Table 4.1: Questions about the validity of scenarios

We separate the statements into three groups based on the scores given. First the statements with which the interviewees largely agree (a score of 4 or 5):

- If the player chooses the best option (the option with the highest score increase) at every node, he / she should have the highest possible score at the end of the scenario. [Question 13]
- It should be possible to get the maximal score. [Question 15]
- For every parameter there should be a path that generates the maximal score. [Question 16]
- Every node (even those with prerequisites) should be reachable in some way. [Question 17]

Then we have the statements with which the interviewees largely disagree (a score of 1 or 2):

- The longest path should yield the highest score. [Question 12]
- The longest path should yield the lowest score. [Question 19]
- Parameters are always positive. [Question 26]

Many interviewees indicated that if the longest path does yield the highest score, it often is a sign of bad scenario design. At the same time they do not think this is necessarily always the case, which might explain why they did not agree with the statement that the longest path should yield the lowest score either.

Lastly, we have the statements with which the inerviewees neither agree nor disagree (a score between 2 and 4):

- The sequence in which subjects have been dealt with, should not matter for the end score. [Question 14]
- Parameters should have no correlations with each other. [Question 18]
- It should be possible to get the minimal score [Question 20]
- For every parameter, there should be a path that generates the minimal score [Question 21]

- The player should sometimes sacrifice a scoring opportunity to receive a better opportunity later on. [Question 22]
- Switching from subject is never a good idea. [Question 23]
- All the (scoring) parameters should use the same scale (1 – 10, 0 – 100 etcetera). [Question 24]
- It should be possible to score above 5% on every parameter. [Question 25]

We already discussed how many of the interviewees found it difficult to talk in absolute statements about the validity of a scenario. This can also be seen in the given scores. The statement "It should be possible to get the maximal score" scores higher (score: 4.8), then the statement "For every parameter, there should be a path that generates the maximal score" (score: 4.2). This seems weird, because the former statement can only be true if the later statement is also true. Interviewee 1 also gives a score of 4 to both the statement that "always choosing the best option should result in the highest score" and the statement that "players should sometimes sacrifice a scoring opportunity to receive a better opportunity down the line". However, these statements are each other's opposite, and can therefore not be true at the same time.

These discrepancies can partly have arisen because our subjects did not understand the question properly. Quotes like "I could imagine a situation, where this is true", seem to indicate, however, that while they broadly agree with certain statements, they want to keep the option to deviate from them. This shows that a scenario author should be able to choose to ignore the recommendations of a tool. Scenario authors should also be able to make an informed choice about whether or not they want to use any potential validation tool for a specific scenario. It should therefore be clear how such a tool works, and what exactly is measured by the tool.

There are two different ways in which the interviewees look at scoring within scenarios. Some interviewees seem to view the game as an assessment, even though none of them ever used it in this way. They often say that players should be able to reach the minimum and maximum score, and that parame-

ters should not have any correlation with each other. These are assumptions that are also often used to check the validity of traditional assessments.

Other interviewees disagree though. In their opinion, the reachability of the minimum and maximum scores, tell us very little about the validity of a scenario. They also voiced opinions like: "scoring is not that important for my scenarios" and "I use scoring mostly to make people aware of what is happening, not to pass a judgement". On further questioning, they often indicated that they do see how certain statements can be useful for describing a scenario, but they do not see the statements as a way to assess validity. In the words of one interviewee: "As a scenario author, you need to understand what you are doing, but correlation can also help you to refine small differences in the effect of their (read: the player's) choices."

When asked which statements they found difficult to check in their own scenarios we got many different answers. Some answers kept reappearing though. Many interviewees found it difficult to check if the best (read: the preferred) path also yielded the highest score. They also wanted to know what the current maximums of their parameters were.

4.3.3 Optimal Path

We continued the interview by discussing the optimal path. We define an optimal path as the path through the tree that generates the highest possible score. In theory an optimal path should be the same as the preferred path of the scenario author. The implicit assumption here is that the player should value high scores, and that a high score should therefore be the reward for good behaviour. Communicate displays the total score in a prominent way, and this supports our implicit assumption. Communicate scenario authors often distribute the scenario themself. In the instructions they provide to the students, they can specify other goals than maximising the total score. This could in turn invalidate our assumptions about the preferred and optimal path. By discussing the optimal path with the interviewee we wanted to find out if they think that maximising the total score is indeed the most important goal of players, or if our definition of the optimal path should be changed.

We again asked the interviewee to assess a couple of statements with a number between 1 and 5, where a 1 means they completely disagree and a 5 means they completely agree. Question 35 is a bit different. In general, school grades can be split into two categories, passing grades and non-passing grades. A threshold (usually 5 or 6) is chosen to differentiate between the two categories. If a scenario author views her scenario as a test for the students, it would make sense if such a threshold also exists for the scoring parameters. For instance, imagine a situation where a player needs to maximize his/her weighted average, but also keep all the individual parameters above a certain threshold. For question 35 we took the definition of the optimal path that was preferred by that individual interviewee, and asked them if they thought that their preferred definition would improve if we added the amendment that the player should also keep all the parameter scores above a certain threshold. The results can be found in 4.2.

Question	I	II	III	IV	V	VI	Total
29	4	4	3	1	4	5	3.5
30	1	2	4	2	4	1	2.3
31	4	4	5	5	5	3	4.3
32	1	1	1	1	2	1	1.2
33	1	1	1	1	3	1	1.3
34	1	1	1	1	1	1	1.0
35	No	No	Yes	Yes	No	Yes	

Table 4.2: Questions about the optimal path

In summary, we have one definition for the optimal path that is broadly seen as useful:

- The optimal path is the path with the highest weighted average of all the used parameters [Question 31]

Three definitions, which the interviewees are uncertain about:

- The optimal path is the path with the highest average over all the parameters (no weights) [Question 29]

- The optimal path is the path that includes the highest score for any individual parameter [Question 30]
- The definition of the optimal path would improve if we added the clause that all the parameters should be above a predefined minimum. [Question 35]

And three definitions that are seen as not useful:

- The optimal path is the path with the minimal number of nodes [Question 32]
- The optimal path is the path that has the highest score on its lowest parameter [Question 33]
- The optimal path is the path with the maximal number of nodes [Question 34]

Definition 31 ("The optimal path is the highest weighted average of all the current parameters") is clearly the definition the interviewees most prefer. This is the same definition that is currently used to calculate the total score in Communicate. The total score is prominently displayed on the feedback screen at the end of every game. It is possible that the interviewees favour definition 31 because of its current prominent use, and that scenario authors of other serious games will favour different definitions. Since we focus our study on Communicate specifically, we will ignore this possibility, and assume that scenario authors favour definition 31.

Many interviewees noted that there is no definition for the optimal path that is applicable in every situation. The total score is the default measure to represent player performance, but other measures might be possible. An example that was given by one of the interviewees was a personality test, where the player is not judged on his/her final score. Instead, the individual scores of parameters would represent the personality of the player. In such a case the definition of the optimal path should be changed, but only for that particular scenario. It would, therefore, be nice if a scenario author can indicate which definition should be used. If this is not possible, the scenario author should at least know the definition used.

4.3.4 Proposed Improvements

In the last part of the interview, we asked the interviewees about their opinion of possible improvements to Communicate. We told them that they should feel free to make their answers as long or short as they wanted. Thus we can not condense the answers into absolute numbers. This makes comparing answers hard. We grouped the questions into four categories, which we will discuss in turn. The exact questions can be found in appendix 1. At the start of each section, we also give a short overview of the different questions.

Showing Scores

The first group contains question 37, 38 and 39. These questions are about displaying the score changes directly in the tree. In question 37 we propose to display all the parameter changes within the DAG itself. Question 39 follows the same idea, but displays only the total score change, and not all the individual parameter changes. In question 39 we asked the interviewees their opinion on color coding the nodes on their netto total score changes. The proposed improvements differ mostly on how much information we display at the same time. In this way, we hope to find out if the scenario authors see information overload as a problem.

The reactions of the interviewees ranged from lukewarm to positive. All the interviewees thought that displaying the score change, would improve a scenario, but they questioned how big this improvement would be. The following quote expresses this opinion: "It certainly offers something, but I do not know how much it would actually help me personally". The interviewees seem to have two primary concerns on the usefulness of the proposed improvements. Will the improvements lead to information overload, and how usefulness is the total score in practice.

Roughly half of the interviewees mentioned the problem of information overload. They did seem to think that this problem can be solved. Many supported the idea that you should be able to customize what changes are visible at any given time. For instance, they proposed an option to select which parameters should be visible at any given time. The color coding system was

most popular, with every interviewee indicating that they liked the idea.

The other problem with the improvements proposed in our questions, is the usefulness of the total score. Most interviewees did think that the total score is important, but they said that they usually do not consider it much while creating scenarios. They see it more as something they need to check after the rest of the scenario is created, while the individual parameters are more of a concern during scenario creation. Many interviewees would therefore rather see that the proposed improvements do not show the changes in the total score, but show the changes in individual parameters instead. Despite this preference most interviewees indicated that the changes in total score were still of some use. A few interviewees indicated that only displaying the total scores would have no use for them at all.

Showing Optimal Path

The next group of questions (40, 41 and 42) all relate to showing the optimal path while editing. We started with asking if the interviewees would like to see the optimal path within one subject. In the next question we extended this to the optimal path in the entire tree. In the last question we proposed an improvement, that shows the optimal path starting from a node selected by the user. Of course, these improvements depend on the definition of the optimal path. This definition is not necessarily the same for every possible scenario, which was something that the interviewees often mentioned.

Despite the problems with the definition of the optimal path, the general reaction to our proposed improvements was very positive. We heard multiple stories of people who were now calculating the optimal paths by hand or who tried to visually arrange the tree in such a way that they could keep track of the optimal path. The interviewees mostly indicated two motivations for this:

First of all, they noted that it was difficult to return to working on a scenario when they had not worked on it for a while. The visual cues helped them remember their original ideas. This problem cannot only be solved by our proposed improvements, but also by adding an option to annotate the optimal path by hand. Although we should note that there is an important difference

between the numerical optimal path and the perceived best path, which we call the preferred path. This problem seems to partially exist because scenario authors have trouble quickly interpreting large trees. Improving on the automatic tree arrangement might help alleviate this problem.

The second motivation for showing the optimal path is that the users can see the maximal score this way. Later in the interview, we bring up the option to show the maximum score ourselves. However, some interviewees brought this up themselves before we had mentioned it, which underlines how helpful this feature could be.

Most interviewees prefer to see the optimal for the entire tree, but they understand that this might be very hard to implement. They indicated that showing the optimal path within one subject, instead of in the entire tree, would still be of some help. A few interviewees even told us they prefer to only see the optimal path within one subject because it would give them simpler and clearer feedback.

One thing all the interviewees agree on is that the option proposed in question 42 (a button, which, when clicked, shows the optimal path from a specified node) is the most desirable of the three options because it would allow them to not only see the optimal path, but also the paths that become optimal, once a certain mistake has been made. This was something that was mentioned quite a lot, not only with this question, but also on other questions. Almost all the scenario authors want players to have the option to correct for a mistake. This means that there need to be semi-optimal paths, which allow a few errors, but still yield a high score. So while the scenario authors see the optimal path as useful, they think that ranking all the different paths on desirability would be even more useful.

We asked the interviewees about their opinion on two usability factors. First there is the calculation time, the time that the computer would need to recalculate the provided information, once the scenario author has made a change in the scenario. Secondly, there is the margin of error. With margin of error we mean both calculations that do not always yield the correct answer and calculations that ignore certain factors of the scenario design, like for instance preconditions. The interviewees told us that the time needed to calculate any provided information is not that important. They think that

a couple minutes of calculation time is still tolerable. They are a lot stricter on the margin of error though. Only one interviewee told us she would be okay if the certainty was less than 100%. Ignoring preconditions might be acceptable, but it would severely limit the usability in their opinion.

Automatic Validation

Question 43, 44 and 45 are about automatic validation of scenarios. In question 43 we discuss validation, where a validator selects a random node, generates the optimal path from that node, and asks a scenario author to validate this path. The improvements proposed in questions 44 and 45 are a bit simpler, because they require less input from a scenario author. A scenario author simply marks some nodes as desirable, in question 44, or undesirable, in question 45. The drawback of this approach is that validation becomes less robust.

At first many interviewees seemed confused about what we meant with these questions, but after some more explanation they all came around and said that they would really like such a function. This emphasizes the fact that any validation tool should be very clear in its functionality and its possible uses. While such a validation tool apparently has some worth, this worth is not necessarily self explanatory. The risk exists that scenario authors will simply ignore the validation option, because they do not understand how or why they should use the option. Clear instruction and a good interface, are of course always important, but they are especially required in this case.

In general, the interviewees liked the option to color code the nodes themselves more than the validating random paths option. This is not completely surprising because these functions seem to align closely with how the interviewee are currently creating scenarios. Almost all interviewees indicated that they usually have a perfect path in mind while creating scenarios. They often even start with creating this optimal path. One interviewee even asked for a color coding function, but without the validation option, before we suggested the option ourselves. He motivated his proposal by stating that color coding would help him keep a better overview while creating scenarios.

The choice between indicating correct nodes or incorrect nodes seems to be a

toss-up. Many interviewees seem to prefer marking the nodes they see as the correct or best option, because this aligns better with how they are currently working. At the same time, they often see more use in marking incorrect nodes. A combination between the two options might be preferable.

Interestingly, two interviewees independently proposed the same new idea. They told us they would like it if the process works the other way around. The editor should calculate all the paths and then rank them on their score. This ranking would be displayed using different gradients of green and red. The scenario author can then quickly assess if the ranking is correct.

In our questions, we introduced the different improvements as calculations that would constantly be refreshed in the background. Later we asked the interviewee if it was a problem if they needed to click on a button to start or refresh the calculation. The interviewees all said that this was not a problem, and quite a few even preferred the second option. We also asked if they would rather see a validation that would encompass the entire scenario or just one subject. Once again there was not one prevailing choice.

Statistics

Lastly, we asked the interviewees if they were interested in seeing general statistics of their scenarios (question 46 and 47). The reactions were mixed. Everybody would very much like to know the maximal score that can be obtained in a scenario. They often even stated this right at the start of the interview as a major problem of the current version of Communicate. Their interest in other kinds of statistics was only lukewarm though. The interviewees had some interest in the minimal score, and a few interviewees were interested in the correlation between parameters, but there was no overwhelming interest.

4.4 Conclusion

The most interesting finding of the interviews is the importance of the optimal path. We define scenario smells by specifying what makes a scenario correct. Quite a few scenario smells relate to the preferred and/or optimal path, and the interviewees seem to mostly agree with the fact that those paths should be the same in a correct scenario.

The importance of the optimal path can be seen in the reactions the interviewees gave on the multiple proposed improvements of communicate. The most preferred improvements all relate to different paths, both optimal and suboptimal, in the dialogue tree. Mathematically, calculating the optimal path, the least optimal path, and a sub-optimal path, are somewhat related. Finding out whether or not it is possible to calculate the optimal path, and if so, how this can be done, therefore also provides us with more information about how we can calculate other paths.

Calculating the maximum score of a parameter is roughly the same problem as calculating the optimal path. It would also be relatively easy for scenario authors to find the maximal score themselves if they knew the optimal path. Combine this with the lackluster interest in such a feature and we propose that statistics should be ignored for now and that the focus should be on calculating the optimal path.

The question whether or not the optimal path is computable is probably not answerable with a strict true or false. It is very well possible that the calculation is possible, but either takes a very long time or will require us to sacrifice some accuracy. We, therefore, wanted to find out which of those two would be the bigger problem for scenario authors.

In general, the interviewees indicate that they find accuracy more important than calculation time. Quite a few interviewees told us that calculation time does not matter at all. It is questionable if this is indeed the case if a scenario authors is confronted with long calculation times, but for now we can conclude that accuracy is more important than calculation time.

The accuracy requirement can itself be split into two sub-requirements. Does

an algorithm provide the correct answer, and does an algorithm take all the information into account? The interviewees were very strict on the correctness requirement, with almost all of them indicating that provided measurements and paths should be correct. The completeness requirement is a lot less strict. The interviewees are willing to accept an algorithm that only works on one subject, and some interviewees are even willing to accept an algorithm that ignores preconditions.

One last observation we have drawn from the interviews is how much the requirements differ from user to user. Even when a proposed improvement was well liked by the interviewees, they often still wanted the option to turn it of. It is important that a user understands what a possible improvement does, and that they can choose to turn it off, or at least ignore its effect. It would be even better if a scenario author has a degree of control over the functioning of an improvement. For instance, by selecting which definition of optimal path an algorithm should use.

In conclusion, we think it is important to find out how the optimal path through a dialog tree can be calculated. It will be useful to compare different methods and their limitations. This in turn can give us some insight into which of the improvements can be realistically incorporated into Communicate, or what changes need to be made to Communicate to allow for said improvements. Although other information beside the optimal path is desirable, we think that it is best to focus on the optimal path first. The interviews we conducted support this premise, so in the following sections, we take a look at different possible implementations of algorithms to find the optimal path. We will then compare their advantages and disadvantages.

4.5 Scenario Smells

In chapter 3 we discussed the possibility of using Scenario Smells to assist scenario authors with creating scenarios. If we take the outcome of the interviews we can create a first set of these scenario smells. It is important that the absence of scenario smells aligns with a correct scenario in the eye of scenario authors. We found the following scenario smells:

- Always choosing the optimal option does not result in the maximum score
- A longest path does yield the maximum (or minimum) score
- A shortest path does yield the maximum(or minimum) score
- The maximum and minimum score are unobtainable
- Two parameter have a high correlation
- The sign of the correlation of two parameters does not match expectations
- A scenario is not subject monotone

Chapter 5

Implementation

In the previous section, we defined a list of scenario smells. If we compare these scenario smells, it becomes clear that many of them reference specific paths through the scenario. These paths are often optimized for a specific requirement. We use phrases like "the shortest path" and "the best path". We call such a path an optimal path. The calculations for these optimal paths are often variations of each other. Calculating the optimal path for length (shortest or longest path) is a variation of calculating the optimal path for overall score (best path).

We created an application that can find scenario smells by examining and comparing different optimal paths. In this chapter, we discuss this application and our reasoning behind the choices we made. We start by discussing how we can calculate an optimal path. Secondly, we discuss how different scenario aspects can influence the necessary calculation time. Thirdly, we will discuss how we translate the different optimal paths into scenario smells. Lastly, we will discuss how we can present the scenario smells found to the scenario author.

5.1 Calculating the Optimal Path

An optimal path is a path through a scenario that returns a highest or lowest score on specific parameters. These parameters can be scenario author defined parameters or more abstract parameters, like the number of nodes. Finding an optimal path is a kind of pathfinding algorithm. For efficiency reasons, many pathfinding algorithms use a heuristic. Examples of this kind of algorithms are algorithms like Dijkstra and a* [10, 11]. A heuristic is a function that ranks possible paths in order of likelihood of being the desired solution. A greedy search algorithm can then be used to find a solution relatively fast.

When we use a heuristic, we need to make a tradeoff between accuracy and speed. For some search problems it is possible to find a heuristic that always results in a correct answer. In Communicate every node can contain an addition or subtraction to a parameter. Communicate also allows nodes to set parameters to a specific value. This means that the value of a parameter in a node, gives no information about the possible values of that parameter in the child nodes. If we cannot make accurate predictions about the parameter values of child nodes, it is impossible to create a heuristic that always results in a correct answer.

Applying Dijkstra's algorithm [10] to a Communicate scenario gives an example of this problem with heuristics. Dijkstra works with a DAG, where every node can increase a parameter value. The algorithm is used to find the path with the lowest parameter value. Dijkstra works with the heuristic that the parameter value of child nodes will be equal or greater than that of its parent node. We can therefore use a greedy search algorithm that always explores the unexplored node with the lowest parameter value. When we reach an end node in the DAG, we can safely assume that the found path is the path with the lowest parameter value, because all unexplored child nodes have a parameter value that is greater or equal to the found end node. In Communicate a node can also decrease a parameter value. This invalidates the heuristic of Dijkstra's algorithm. We can no longer make correct assumptions about the parameter values of the unexplored child nodes.

Because of the problems with finding a heuristic, and because accuracy is

highly desired by scenario authors, we decided to implement a brute-force method that traverses all paths. We then compare all paths to find the optimal paths. This is computer resource intensive, but also ensures the validity of the answer.

5.2 Size of Search Space

In the last section, we discussed our algorithm for finding an optimal path. The total amount of possible paths greatly influences the calculation time. In this section we discuss how we can calculate the total amount of possible paths through a scenario. This will help us compare how different scenario components influence the complexity of the optimization problem.

5.2.1 Basic Combinators

A Communicate scenario consists of a collection of statement combinators. These combinators must be traversed in certain specific orders. We can represent this collection of combinators and their possible orders in a diagram. An example of such a diagram can be found in 5.2. Such graphical representations are helpful for understanding small dialog trees, but the images for actual Communicate scenarios can be quite large and hard to interpret. These diagrams also give us little information about the amount of possible paths in a scenario. We, therefore, need an easy way to define a scenario in a completely textual way.

The most basic connection in a scenario is a sequential combinator. One statement is followed by another statement. We denote a sequential combinator with the <*> symbol. If we want to state that statement **N1** is always followed by statement **N2**, we write this as **N1** <*> **N2**. In Communicate a statement is often not followed by one specific other statement. Instead there exist multiple possible sequential connections, between which a player must make an exclusive choice. We denote an exclusive choice with the <|> symbol. If statement **N1** can be followed by either **N2** or **N3**, we write this as **N1** <*> **(N2** <|> **N3)**.

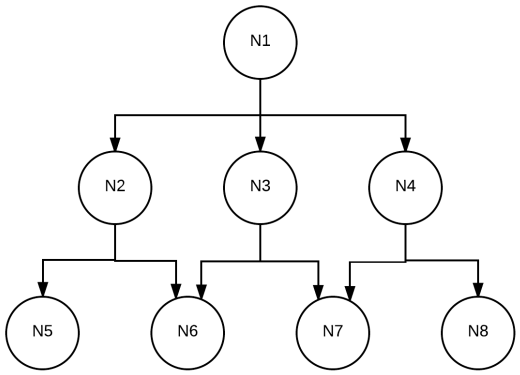

Figure 5.1: Node tree

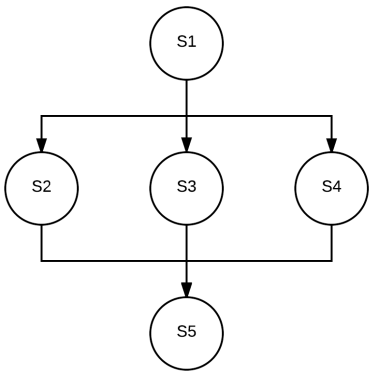

Figure 5.2: Subject tree

Besides the exclusive choice, there is also the parallel choice. A parallel choice combinator connects multiple statements that all need to be traversed but can be traversed in any other. The symbol for parallel choice is <||>. Parallel choices are not necessary for defining a scenario, but they do improve readability. For example, if we have a statement N1 that is followed by **N2** and then **N3**, or first by **N3** and then **N2**, we can write this as **N1** <*> **((N2** <*> **N3)** <|> **(N3** <*> **N2))**. However, the description is easier to understand if we write **N1** <*> **(N2** <||> **N3)**.

We can use this notation to define an example scenario. In figure 5.2 we display an example subject structure. Every subject has a DAG similar to that of figure 5.1. In our notation the entire scenario looks like this:

S1 <*> **(S2** <*II*> **S3)** <*> **S4**
S1 = N1 <*> (
(N2 <*> **(N5** <*I*> **N6))** <*I*>
(N3 <*> **(N6** <*I*> **N7))** <*I*>
(N4 <*> **(N7** <*I*> **N8))** <*I*>
)
S2 = N9 <*> (
(N10 <*> **(N13** <*I*> **N14))** <*I*>
(N11 <*> **(N14** <*I*> **N15))** <*I*>
(N12 <*> **(N15** <*I*> **N16))** <*I*>
)

S3 = N17 <*> (
(N18 <*> (N21 <I> N22)) <I>
(N19 <*> (N22 <I> N23)) <I>
(N20 <*> (N23 <I> N24)) <I>
)
S4 = N25 <*> (
(N26 <*> (N29 <I> N30)) <I>
(N27 <*> (N30 <I> N31)) <I>
(N28 <*> (N31 <I> N32)) <I>
)

We can use the three defined combinators to calculate the total amount of paths in a scenario. The total amount of paths of a sequential combinator, for instance, is the product of the total amount of paths on the left and right side of the connection. If there are 4 possible paths through **S1** and 6 possible paths through **S2**, there are 24 possible paths through **S1** <*> **S2**. We calculate the amount of paths for an exclusive choice and sequential combinator with the following formulas:

amountOfPaths (x <*> y) = amountOfPaths(x) * amountOfPaths(y)
amountOfPaths (x <I> y) = amountOfPaths(x) + amountOfPaths(y)
amountOfPaths(1 node) = 1

Where both x and y are subjects or (groups of) nodes.

For calculating the total amount of paths of a parallel combinator, there are three important facts we need to consider. Firstly, as described before every parallel combinator can be rewritten as an exclusive choice between sequential combinators. For instance, **S1** <||> **S2** can be rewritten as **(S1** <*> **S2)** <|> **(S2** <*> **S1)**. Secondly, sequential connections are evaluated by taking the product of the left and right element. Thirdly, when we take the product of a sequence of numbers, the order in we multiply them does not matter. This means that the order in which we sequentially connect different elements does not change the total amount of paths. **S1** <*> **S2** has the same number of paths as **S2** <*> **S1**. We can combine these three facts, to determine the total amount of paths in a parallel connection. First, we

calculate how many paths there would be if all the elements were sequentially connected. Then, we multiply that answer by the number of ways that the elements can be sequentially connected.

When calculating the total amount of paths in a scenario, we can evaluate every sequential and exclusive choice combinator on it own. This will not work for a parallel combinator. We need to know exactly how many elements are connected in parallel to calculate how many different ways there are to sequentially connect the elements. We, therefore, need to evaluate all the elements combined in parallel at the same time. We can do so with the following formula: number of elements! * amount of paths if sequentially connected. For example, take the scenario **S1** <||> **S2** <||> **S3**, where **S1** has 4 possible paths, **S2** has 6 possible paths, and **S3** has 8 possible paths. If we combine the subjects sequentially there are 192 (4*6*8) paths. In the entire scenario, there are 1152 (3! * 192) possible paths.

We can use the introduced formulas to calculate the total amount of paths through our example scenario. We start by calculating the total amount of paths through **S1**. If we replace all the connections with their mathematical formula we get: 1 * (1 * (1+1) + 1 * (1+1) + 1 * (1+1)) = 6. In our example scenario, the DAG structure for every subject is the same. All the subjects, therefore, have 6 possible paths. On the subject level, three elements are connected in a sequential order. We already evaluated **S1** and **S4**, but still need to calculate the number of paths in **S2** <||> **S3**. As described in the last paragraph the number of paths is equal to 2! * (6 * 6) = 72. This means that the total amount of paths for the entire scenario is 6 * 72 * 6 = 2592.

Based on these calculations, we can order the different combinators on their impact on the total amount of paths. Parallel combinators have the largest footprint because they introduce a factor into the calculation. In our example scenario, for instance, the one parallel connection that is included doubles the total amount of possible paths. Sequential combinators use multiplication and have the next largest footprint. Exclusive choice combinators use addition and have the lowest footprint.

5.2.2 Interleave Points

Interleave Points are statements where the user can switch to another subject that is connected in parallel with the current subject. The remaining part of the current subjects will still need to be traversed before the user can move on to the next sequential subject. In our notation, we denote that a statement is an interleave point by placing an exclamation mark after the name of the statement. This notation keeps the scenario definition somewhat compact. It is possible to give a definition of a scenario with an interleave point, without using the interleave notation. We will sometimes use that notation to make the calculations more clear.

To illustrate our points about interleave points, we will make use of a slightly altered version of our previous example. The new definition for **S2** looks like this:

S2 = N9 <*> (
(N10! <*> (N13 <|> N14)) <|>
(N11 <*> (N14 <|> N15)) <|>
(N12 <*> (N15 <|> N16)) <|>
)

As can be seen in the definition, we marked statement N10 as a interleave point. The rest of the scenario definition remains unchanged from our previous example.

If there are no interleave points a player needs to finish one subject, before starting the next. This means that there exists no possible path where the nodes of one subject are separated by other nodes. The existence of interleave points changes this. Even then, there are groups of nodes that always follow on each other, without the intrusion of nodes from other subjects. We call such groups subject atoms. Every interleave point splits one subject atom into three distinct subject atoms. The first subject atom contains all the nodes from which the interleave point can still be reached. The second subject atom contains all the nodes after the interleave point. The third subject atom contains all the nodes from which the interleave point cannot be reached. For instance, **S2** can be separated into 3 subject atoms. The

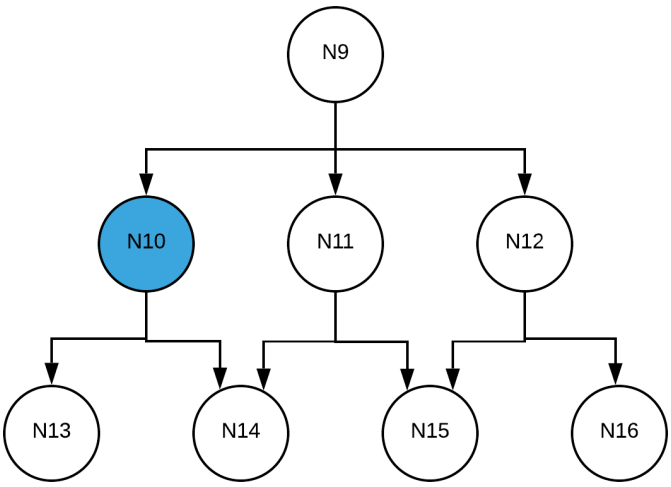

Figure 5.3: Node tree with interleave point

first atom (**A1**) consists of **N9** only. The second atom (**A2**) consists of **N13** and **N14**. The third atom (**A3**) consists of **N11**, **N12**, **N14**, **N15** and **N16**. Notice that **N14** is in two subject atoms.

The separation of a subject into atoms is useful for calculating the total amount of paths. We can now separately calculate how many paths do not include the interleave point, and how many paths do include the interleave point. In the first situation, we can combine the atom before the interleave point and the atom where no interleave point can be reached into a dummy subject. In this dummy subject, the statement with the interleave point and the paths leading to it are nonexistent. This dummy subject is connected to other subjects in the same way as the original subject, and the total amount of paths can be calculated via the established methods.

We will now explore how we can calculate the total amount of paths that include the interleave point statement. As discussed before, we separated the subject into two subject atoms. Atom 1 (**A1**) contains all the statements up to and including the interleave point. Atom 2 (**A2**) contains all the statements after the interleave point. The player can traverse atom 1, atom 2, and all subjects connected in parallel, in any order as long as atom 1 is

traversed before atom 2. Like with a normal parallel connection, we want to find out how many sequences are possible for traversing all the different subject atoms and subjects. We can accomplish this by writing a scenario definition that does not use the interleave point notation. We illustrate this with the **S2** <||> **S3** part of our example scenario. The scenario definition without the interleave point looks like this:

(A1 <*> **(A2** <||> **S3))**
<|>
(S3 <*> **A1** <*> **A2)**

With our earlier formulas, we can calculate the total amount of sequences as $2! + 1 = 3$. We can also calculate the amount of possible paths if **S2** and **S3** were sequentially connected, by using the formulas from the last section. We then multiply the number of subject sequences with the number of possible paths if the subjects were sequentially connected, to get our total amount of paths through the parallel connection.

Writing a scenario definition without the interleave point notation, can become more difficult in larger scenarios. For instance, consider the following scenario:

S2 <||> **S3** <||> **S4**

The scenario still includes one interleave point in S2. The scenario definition without the interleave point looks as follows:

(A1 <*> **(A2** <||> **S2** <||> **S3))**
<|>
(S2 <*> **A1** <*> **(A2** <||> **S3))**
<|>
(S2 <*> **S3** <*> **A1** <*> **A2)**
<|>
(S3 <*> **A1** <*> **(A2** <||> **S2))**
<|>
(S3 <*> **S2** <*> **A1** <*> **A2)**

With this definition we can calculate that there are 3! + 2! + 1 + 2! + 1 = 12 possible subject sequences.

If there are four parallel subjects and one interleave point there are 60 possible subject sequences. The calculation can be written as 4! + 3 * (3! + 2 * (2! + 1)). The calculation for the amount of subjects sequences with 3 parallel subjects, can be written as 3! + 2 * (2! + 1). There is a pattern to the different calculations. We can use the following formulas to calculate the amount of paths in a scenario with one interleave point:

$$\text{AmountOfSequences}(0) = 0$$
$$\text{AmountOfSequences}(x) = x! + (x\text{-}1) * \text{AmountOfSequences}(x - 1)$$
$$\text{AmountOfPaths} = \text{AmountOfSequences} * \text{PathsInSequentialConnection}$$

Where x is the amount of subjects.

We now have enough information to calculate the total amount of paths for a parallel combination with one interleave point. First, we separate the subject with the interleave point into three subject atoms. Second, we calculate the total amount of paths that do not include the interleave point by using a dummy subject. Third, we calculate the total amount of sequences that do include an interleave point with the pseudocode. Fourth we multiply the number of sequences by the amount of paths in any one sequence to get the total amount of paths with the interleave point. Lastly, we add the amount of paths without the interleave point together with the amount of paths with an interleave point.

In this section we only discussed how to calculate the total amount of paths if there is one interleave point in a parallel combination. The exact calculation will change if there are multiple interleave points. However, the described method can be used to create new formulas for different amounts of interleave points.

The impact of interleave points on the total amount of paths is quite high. With every interleave point an extra subject atom is added to the parallel combination. This means that adding an interleave point will have almost the same effect as adding an extra subject. The actual impact is a little lower, because the separation into atoms will reduce the subject complexity.

However, adding an interleave point is a lot less work for the scenario author than adding a new subject. This means that interleave points can quickly become the largest contributor to the total amount of paths.

5.3 Subject Monotonicity

Scenarios are often built over a period of time. Scenario writers often create an initial scenario within a short time frame and then make small incremental changes each time the scenario is used. In our interviews, the interviewees noted that returning to an earlier created scenario can be difficult. It is hard to remember good or bad dialogue paths over long periods of time, and rereading the entire scenario is a lengthy process. This process might become easier if we can separate the scenario into smaller autonomous parts.

We can use monotonicity to decide which parts of a scenario can be considered autonomous. Our use of monotonicity is based on mathematical monotonic functions. To determine if a function is a monotone function we look at any two inputs, x and y, and their outputs, f(x) and f(y). If $x \leq y$ always means that $f(x) \leq f(y)$ we call the function monotonically increasing. Likewise, for a monotonically decreasing function we have that for all x and y, if $x \leq y$ then $f(x) \geq f(y)$. There are multiple different ways to define monotonicity within Communicate depending on the chosen input and output. We will now discuss the three definitions we deem the most important.

For the first method, we can take the value of all the individual parameters at the end of the scenario as input. The output is the total score at the end of the scenario. In that case, a scenario is monotone if an increase of one parameter's end score leads to an equal or greater total end score. Likewise, a decrease in one parameter should lead to an equal or lesser end score. We call a scenario with this kind of monotonicity parameter monotone.

For the second kind of monotonicity, we take all the individual parameter changes as input and the parameter values at the end of a scenario as output. The scenario is monotone if an increase of a parameter change on one node leads to a parameter end score equal to or greater than the previous end score. Likewise, if we decrease a parameter change, the end score should be

equal or less than the previous end score. For example, take a scenario with a parameter A. There are three nodes that change the value of A. The changes are +3, +5 and +9. If we alter the first parameter change from +3 to +4, the end value of A should be equal or greater than before. In this kind of scenarios, we can change individual nodes without considering the rest of the scenario. We, therefore, call this kind of scenario node monotone.

The third approach is somewhat similar to the node monotone approach. In this case, we take the set of net parameter changes of a subject as input. For instance, imagine a subject that only has two possible paths. The first path has a parameter increase of 10 and a parameter decrease of 5. The net value of this path would be +5. The second path has a parameter increase of 10 and a parameter increase of 2. The net value of this path would be +12. The set of net parameter changes would be +5, +12. The output of our monotonicity function would be the parameter score at the end of the scenario. An increase in the net change in one subject should mean that the end value of the parameter is equal or greater than before. A decrease in the net change in one subject should mean that the end value is lower or equal than before. We call a scenario with this kind of monotonicity subject monotone.

Let us further illustrate subject monotonicity with an example. Imagine a subject that has one parameter, called A. There are 4 paths through the subject. Parameter A has a start value of 10. The four different end scores are 5, 15, 15 and 20. The set of net score changes for this subject is -5, +5, +5, +10. If we increase one of the individual changes to A, all the values in the set should also increase or stay the same. After the score change, the new set of net score changes is -5, -5, +5, +10. This means that one of the scores in the set has decreased. The subject is not monotone.

Of the three kinds of monotonicity, we consider subject monotonicity to be the most important. If a scenario is subject monotone, we can safely consider the subjects to be independent. We can make alterations to one subject, and assume that the entire scenario will be effected in a similar fashion. For instance, if we increase one of the parameter changes in a subject, we know that the end value of that parameter will also increase, or at least stay the same.

It is useful for scenario authors to know if a scenario is subject monotone. There is nothing inherently wrong with a scenario that is not subject monotone. In fact, there are many possible cases where subject monotonicity is undesirable. For the inexperienced scenario writer, subject monotonicity might still be a desired attribute. It is quite common for scenario writers to make small changes to a scenario, after a test in a practical environment. A scenario writer can quite easily miss how a change in one subject, has unintended consequences for the gameplay or scoring in another subject. Knowing that a subject can be changed without altering the other subjects, can be reassuring to novice scenario writers.

Subject monotonicity also is a very useful property for calculating optimal paths. If a scenario is subject monotone, we can consider each subject individually. We can calculate an optimal path for every subject and concatenate them together. This approach avoids the growth in the number of paths due to parallel and sequential connections between subjects. It also shortens the length of the paths we need to calculate.

A scenario without preconditions is always subject montone. In our earlier example, we used a scenario with a single parameter called A. One of the subjects in this scenario had the following net score changes: -5, +5, +5, +10. We can increase one of the parameter changes in such a way that the net score changes become: -5, +5, +5, +15. If all the other subjects only contain increases or decreases to this parameter, parameter A should either be 5 higher or the same as before the change. Either the player followed a path that contained the score change, and parameter A will be 5 higher than before, or the player took another path, and his/her score will be equal to before. It is also possible that one of the other subjects contained a set operation. In that case, the final value of A will always be the same, no matter the changes in the earlier subjects. This same thought experiment can be repeated for decreasing parameter A, which means that without preconditions a scenario will always be subject monotone.

The example in the previous paragraph also illustrate why a scenario with preconditions is not subject monotonicity. In the example, we made a change to parameter A in one of the subjects. If one of the other subjects has a precondition based on parameter A, we can no longer be sure that the scenario is monotone. There could be a situation where our change to the

earlier subject, results in the fact that this precondition is no longer met. This means that a part of the latter subject is no longer available. This part of the later subject can also have changes to parameter A. There now exists a change to our example subject, where we increase the value change of parameter A, but the overall score decreases because the player is locked out of a part of a later subject.

Based on the previous examples we can state that a scenario is subject monotone if it has no preconditions. However, this is not be the most useful definition of subject monotony. We want to give the scenario writer new information about how atomic the subjects are. The scenario author usually already knows if a scenario uses preconditions or not. The strict definition of subject monotony therefore provides no new information. A slightly more useful version of subject monotonicity only allows for making alterations to existing parameter changes, not for adding completely new ones. In that case, a scenario with preconditions can still be subject monotone, as long as the parameter used in the precondition is not altered outside of the subject containing the preconditions. This is the definition and measurement we will use for our experiment.

5.4 Calculating Scenario Smells

In chapter 4 we identified the following scenario smells:

- Always choosing the optimal option does not result in the maximum score
- A longest path does yield the maximum (or minimum) score
- A shortest path does yield the maximum(or minimum) score
- The maximum and minimum score are unobtainable
- Two parameter have a high correlation
- The sign of the correlation of two parameters does not match expectations

- A scenario is not subject monotone

As described in section 5.1, the brute-force depth-first search tree is our preferred way of identifying different paths. It is a method that provides a hundred percent accurate answers, and that strikes a balance between the use of work memory and processing power. We can use the depth-first tree to find the paths with a maximal score, the paths with a minimal score, the shortest paths, and the longest paths.

We also want to calculate all the paths in which the player always makes the optimal choice. We define the optimal choice as the connected node with the highest immediate total score increase. To accomplish this we use a greedy depth-first search tree. This greedy tree only considers nodes which are optimal choices. In order to limit calculation time, the program uses a list of nodes that still need to be explored. This is problematic for some scenarios. Mainly scenarios that are either very big or that include a lot of nodes without parameter increases. We, therefore, included an option to ignore the first scenario smell from the scenario smell calculations.

For calculating the correlation we use the Pearson correlation coefficient. To accomplish this we need a list of all the possible end values of each parameter. We can then calculate the Pearson correlation of each two parameters using the following pseudo-code:

```
input: Parameter List one, Parameter List two, List length count
Result: Correlation between two parameters
while i < count do
    totalSum += one[i].Value * two[i].Value ;
    firstSum += one[i].Value ;
    secondSum += two[i].Value ;
    firstSquireSum += one[i].Value * one[i].Value ;
    secondSquireSum += two[i].Value * two[i].Value ;
    i++ ;
end
top = (count * totalSum) - (firstSum * secondSum) ;
bottomA = count * firstSquireSum - Sqrt(firstSum) ;
bottomB = count * secondSquireSum - Sqrt(secondSum) ;
Bottom = Sqrt(bottomA) * Sqrt(bottomB) ;
correlation = top / bottom ;
```

The last smell we calculate is subject monotony. The program uses two two-dimensional bool arrays, where the first index corresponds with a subject id, and the second index corresponds with a parameter. In these bool arrays, we record if a subject has preconditions using a specific parameter. We also record if a subject changes a certain parameter. For each found precondition the program checks if no other subject contains changes to the used parameter. If this is the case the scenario is flagged as subject monotone.

The brute-force trees can find the optimal path for specified criteria. The standard brute-force considers every single path while searching for this optimal path. It is, therefore, possible to perform multiple searches at the same time. In designing our program we needed to make a choice about which search trees where combined. This is in large part a choice between more extensive use of memory or processing power. It is also worth considering that certain scenario smells are more related to each other and are, therefore, more likely to be used at the same time. The last consideration is that we can calculate some scenario smells for each parameter individually, while other scenario smells can only be calculated for the total score. In our final implementation, we combined the calculations for the maximal score, the minimal score, the shortest paths, the longest paths, and subject monotonicity. The correlation calculations are done by a separate search tree.

5.5 Scenario Smells Report

Once the program has finished with calculating the scenario smells, it places the results in plain text files. The text files are combined in a single directory for easy access. The most important file is the main.txt file. In this file, we give an overview of the different scenario smells. In general, we only note if a given scenario smell is present or absent. At this point, we avoid giving exact information. The only exception are the correlations since it is unclear how large a correlation needs to be, to become a scenario smell.

If a scenario writer desires more information, they can take a look at the more specific files. There is a plain text file for each parameter and one for the total score. In these files, we give more information about the found paths for the maximal score, and the found paths for optimal gain at every choice. The report contains the found path, and also the final values of all the parameters, including the total score.

Before the program is executed a scenario writer needs to make a choice between displaying all the paths or only the first found path. Displaying all the found paths, gives more accurate information, but can also make the report harder to understand. This especially a problem for scenarios with a large number of optimal paths. The default option, therefore, is to display only the first found path.

Chapter 6

End Interviews

6.1 Introduction

The goal of the end interviews is twofold. The first goal is to find out how our software performs on actual scenarios. The second goal is to investigate the reaction of scenario authors to our software. We interviewed four scenario writers. We asked each interviewee to send us one of their own scenarios. We used these scenarios in the interviews, and also as a general test of our software. Another two scenario writers were unavailable for interviews but did supply us with scenarios.

In our original plan, we would create a full report for the supplied scenarios. In practice, it became apparent that for some scenarios, this was impossible within a reasonable timeframe. We, therefore, implemented restrictions on the calculation time and computing resources. The creation of the report could at the most use 4GB of working memory. The calculation time was capped at 24 hours. If necessary, we adapted the scenarios, to stay within these bounds. The anonymized scenarios can be found in the appendixes.

We started the interviews by showing the interviewee how the software works. At this point, we also explained any changes to the scenario we had to make. We then showed the interviewee the different functions of our program, and

how the results were presented in the scenario report. During this demonstration, we discussed the results with the interviewee. The focus was on three different aspects:

- Is the result what you expected, and do you think this indicates a problem in your scenario?
- If so, what kind of actions would you take to rectify the situation?
- Are you missing information, required for rectifying the situation?

We then tried to qualify the different scenario smells, by asking the interviewee about their usefulness. Like in the earlier interviews, we asked them to give each smell a rank between 1 and 5. A 1 represents a smell (or calculation) they would never use. A 5 represents a smell they would always use. The goal of this chapter is to see if their earlier opinions have changed now that they have seen the full product.

At the end of the interview, we discussed the full software with the interviewee. We asked them if the liked the software and if they would use it. We also asked them about the calculation time and the required computer resources. Because the interviewees were quite opinionated about the calculation times, we separated these results into a separate chapter.

6.2 Usability of Scenario Smells

The current scenario smell report has a lot of different aspects. We asked the interviewee to rate how likely it is that they would use the different aspects. A 5 represents a function they would definitely use. A 1 represents a function they would definitely not use. The individual answers, as well as the average answer, can be found in the following table.

The option to calculate the best path is clearly the most well-liked function. It has the highest average (5) and the lowest variance (0). The interviewee often emphasized how much they liked this function, by using phrases like “absolutely”, and describing it as “very important”. Earlier we concluded

	I	II	III	IV	Average	Variance
Best Path	5	5	5	5	5	0
Worst Path	2	2	1	5	2.5	2.25
Longest Path	5	4	2	3	3.5	1.25
Shortest Path	2	1	1	3	1.75	0.69
Best Option	3	5	5	5	4.5	0.75
Longest Path	5	4	3	4	4	0.5
Shortest Path	2	1	2	4	2.25	1.19
Min and Max	4	5	5	5	4.75	0.19
Correlation	4	2	5	3	3.5	1.25
Subject Monotone	4	4	2	3	3.25	0.69

Table 6.1: Usability of Scenario Smells

that the best path is an important part of how scenario authors rate scenarios, these interviews confirm this conclusion to be correct.

The least liked functions seem to be the option to calculate the shortest path. It has the lowest average (1.75) and a pretty low variance (0.69). Multiple interviewees told us they often include at least one short, clearly wrong path. This makes calculating the shortest path quite easy. The interviewees often said they already knew the shortest path, and that using the software for this calculation would be a bit redundant.

It is notable that the grading of a few functions has quite a bit of variance. Most notable: calculating the worst path (2.25), calculating the longest path (1.25), matching the shortest path with the best or worst path (1.19), and calculating correlation (1.25). After our earlier interviews, we concluded that scenario authors have quite different approaches to scenario creation. This could be an explanation for the high variance. In general, the interviewees seemed more willing to use the functions that match there current scenario creation process.

The interviewees often described how important it is that a scenario feels correct. Factuality and correct grading are important. However, it is more important that a scenario feels natural and fair. All the interviewees mostly make training scenarios, and the scores seldom have actual consequences. The interviewees, therefore, rate the experience of the player higher, than

the correctness of the scores. This is reflected in how they rate the different functions of the scenario smell report. The option to calculate the best path and the option to calculate the worst path, are equally important if the scores are used for grading. However, the option to calculate the best path is rated as much more important by the interviewee. In the words of one interviewee "a student immediately notices when the maximal score in unobtainable, they probably won't notice if the minimum score is unobtainable."

The interviewees had difficulties understanding some of the aspects of the scenario smell report. Some functions (Best path, Worst path, Longest path, Minimal and maximal score) were very clear, and at first, the interviewees seemed to gravitate towards those functions. They often asked questions about these functions and had ideas about how they could use them to improve their scenarios. In general, the interviewees found it difficult to understand what we meant with "the path where the player always makes the correct choice", and "subject monotonicity". At first many interviewees seemed to, therefore, regard these functions as unimportant. In the end, the interviewees still have a generally favorable opinion about these functions. They often stated that they could see why the functions were important but would have difficulty in actually using these functions to improve their scenario's.

Summarising:

- There are many different ways to work on a scenario. It might be a better approach to evaluate individual scenario smells, instead of a global report.
- The function to calculate the best path and match these with the maximum score are well-liked and easy to understand. They should be the main focus of any actual implementation.
- Scenario authors favor player experience above grading correctness. Their choice of scenario smells reflects this. This should influence how we approach our scenario smell report.
- The usefulness of a scenario smell is somewhat related to how easy it is to understand and make use of it to change the scenario. Finding ways

to give more and better information about some smells, might improve their usefulness.

6.3 Calculation Time

During the development process, we tested the scenario smell calculation with our own test scenarios. The calculation times for these scenarios was a matter of at most a couple of minutes. Our mathematical analysis of scenario structures has shown that this might be different for scenarios that are in actual use. Moreover, we stated we expect the calculation time growth to be worse than exponential. Our tests with the scenarios of our interviewees confirm this statement. We will discuss this in more detail in chapter 7.

The exponential growth of calculation time makes calculation time limits an important discussion point. We asked the interviewees how long the calculation of scenario smells might take. We also asked them if they were more or less likely to use the software if it would remain separate from the actual Communicate application. In their answers the interviewees seemed to consider the following aspects:

- Do I use the scenario smell report to improve a scenario, or to prove the correctness of a scenario?
- Does the resource use of the scenario smell calculation, hinder other tasks of my computer?

For the scenario smell report to be useful in improving the scenario, the calculation time should be relatively short. Most scenario authors prefer to work on one scenario at a time. Their work would largely stay on hold, while the scenario smells are calculated. A calculation time shorter than 15 minutes is preferred, and 1 hour is stated to be the limit. In some cases, the calculation time limit could be stretched to 6 hours, but this is highly dependent on the importance of the scenario and the scenario smell.

Many of the interviewees would also like to use the scenario smell report as proof of correctness. In those cases, they are willing to wait a bit longer on

the results, since they can work on other projects in the meantime. For most scenarios, the calculation time limit would be 24 hours in such cases. Some interviewees were willing to stretch this to a couple of days for important scenarios, but in those cases, the usefulness would be severely limited.

While working on a scenario, the interviewees do not necessarily need a complete scenario smell report. They would prefer to focus on specific smells if this would shorten the calculation times, and therefore improve their workflow of writing the scenario. The opposite is true for using the scenario smell report as proof of correctness. In that case, a complete report is preferred.

The interviewees have no problem with using a program outside of Communicate. Even in its text-based form, they found the test program easy to understand, and would use it for their own scenarios. The interviewees are concerned that the required calculations would put a strain on their system. They would prefer a built-in scenario smell calculation if this would mean that the calculations are done on a server. If the calculations are done on their own computer, the computer should remain usable. The use of working memory should be limited, and they would prefer a calculation time shorter than one night of sleep (8 hours).

Chapter 7

Evaluation

So far we discussed the theoretical possibilities and limitations and the user experience of our demonstration program. One last factor to consider is the practical performance. In this section we will discuss the performance of our demonstration program on some actual, practically used, scenarios. We start with shortly discussing a few average scenarios. Then we will discuss three more extreme scenarios in greater detail

The first test scenario is a scenario we have created ourselves for testing purposes. Communicate scenario authors have created the other scenarios. These scenarios have also seen practical use. We created anonymized versions of these scenarios and included them in our demo code. The structure of the DAG's remain unchained, but we replaced all the text. For the extreme scenarios, we also created multiple versions of the scenario, each one increasing in complexity. The test scenarios, along with the program code, can be found on: https://git.science.uu.nl/T.J.Overbeek/scenario-smells.

For the tests, we used a computer with an Intel Core i7 7700 (Quad core, 3.6 GHz) with DDR4 (1066 MHz) memory. One of the conclusions of our interviews was that the calculation time should stay within specified limits. Depending on the interviewee this calculation time limit Ranges from a couple of hours to 8 hours. We, therefore, decided to limit all of our tests to 24 hours. This limit gives us some extra room but also keeps the test within the realm of actual use. Likewise, we decided to limit the memory use to 4

GB. The interviewee noted that they still would want to use their computer. Since most computers nowadays have between 8 GB and 16 GB of working memory, a limit of 4 GB should assure that this is the case.

7.1 General Scenarios

First, we want to discuss the 7 scenarios for which our program can generate a complete scenario smell rapport within 24 hours. In table 7.1 we listed the most important properties of these 7 scenarios. In table 7.2 we list the calculation time and memory use of a best path calculation for the test scenarios. In table 7.3 we list the calculation time and memory use of a generating a complete scenario rapport.

Name	Number of evaluated parameters	Number of subjects	Number of parallel subjects	Highest Node count in a single subject
Test Scenario 1	2	5	3	20
Test Scenario 2	7	1	0	131
Test Scenario 3	2	4	0	24
Test Scenario 4	7	1	0	132
Test Scenario 5	8	4	2	20
Test Scenario 6	3	2	0	64
Test Scenario 7	4	4	0	72

Table 7.1: Properties of test scenarios

The first 6 test scenarios all show the same general behavior. The calculation time for a complete scenario smell rapport is always well under one minute. The memory use stays generally stays between 20 and 40 MB. The only two exceptions to this rule are scenario 2 and 4. These two scenarios have a somewhat similar structure. The most obvious between these two scenarios and the other test scenarios is the high node count in a single subject. For multiple scenarios, the program has an exact memory use of 23,9 MB when calculating the optimal path. This seems to indicate that for these smaller scenarios the memory use is mostly caused by overhead and that the actual

Name	Calculation Time (ms)	Calculation Time (minutes)	Memory Use
Test Scenario 1	11,164	0.1861	1.1 GB
Test Scenario 2	3,108	0.0518	23.9 MB
Test Scenario 3	150	0.0025	23.9 MB
Test Scenario 4	2,846	0.0474	23.9 MB
Test Scenario 5	34	0.0006	21.6 MB
Test Scenario 6	74	0.0012	23.9 MB
Test Scenario 7	441,487	7.74	42.4

Table 7.2: Best Path Calculation

Name	Calculation Time (ms)	Calculation Time (minutes)	Memory Use
Test Scenario 1	34,550	0.5758	2.2 GB
Test Scenario 2	30,711	0.5119	785.5 MB
Test Scenario 3	400	0.0067	33.2 MB
Test Scenario 4	28,897	0.4816	803.9 MB
Test Scenario 5	78	0.0013	25.2 MB
Test Scenario 6	271	0.0045	24.6 MB
Test Scenario 7	OUT OF MEMORY		

Table 7.3: Complete Scenario Smell Analysis Calculation

scenario has little influence on memory use.

Overall we conclude that for most scenarios it is possible to generate a scenario smell rapport within an acceptable calculation time and memory use. This is even true for scenarios which are a bit more complex, for instance, those with 4 subjects. If the subjects themselves are very complex, meaning they have a high node count, the memory use seems to increase dramatically. Calculation time will also increase, but stay well within acceptable bounds. For three scenarios our program could not generate a scenario smell rapport within the set time and memory limit. In the rest of this chapter, we will discuss these three subjects in more detail.

7.2 A scenario with sequential subjects

Scenario 8 has 6 subjects. All subjects are sequentially connected. Most subjects have reasonably high complexity. The highest node count in a single subject is 58. The lowest node count in a single subject is 16. The scenario is built to allow for replays. If a player makes a grave mistake, he/she will receive feedback and is returned to an earlier part of the dialogue tree. This means that the average length of a path is longer than usual for a scenario of this size.

The goal of this test is to indicate the effect of sequentially connected subjects. We made 6 different test cases. Each test case has one more subject than the previous test case. We always start from the beginning of the scenario. The third test case, for instance, contains the first three subjects of the scenario. Subject 4 and 6 are smaller subjects, otherwise, the subjects are roughly comparable.

Name	Number of subjects	Total number of nodes	Calculation Time (ms)	Calculation Time (minutes)	Memory Use
8A	1	39	9	0.0002	18.5 MB
8B	2	96	8,744	0.1457	23.1 MB
8C	3	154	21,631,454	360.52	123 MB
8D	4	170	OUT OF TIME		

Table 7.4: Best Path Calculation

Name	Number of subjects	Total number of nodes	Calculation Time (ms)	Calculation Time (minutes)	Memory Use
8A	1	39	59	0.001	22 MB
8B	2	96	125,830	2.097	3.6 GB
8C	3	154	OUT OF MEMORY		

Table 7.5: Complete Scenario Smell Analysis Calculation

For the best path calculation time is the limiting factor. The first subjects have a very low calculation time (less than a minute) and a low memory requirement. When we add a third or even fourth subject, both the calculation

time and memory requirement make a significant jump. However, the calculation times growth is a lot higher than that of the memory requirements. The calculation time of the best path in a scenario with three subjects is, according to the interviewees, already problematic. Most interviewees stated that they are willing to wait a few hours, but that a calculation time of more than an hour hinders their productivity. Meanwhile, even with four subjects, the memory requirements are well within acceptable bounds. Memory becomes more of a problem with the complete rapport. With two subjects the memory requirement is already up to 3,6 GB. The likely culprit is the "always best option" tree, which is relatively memory intensive.

We conclude that a "best path" calculation for scenarios with multiple sequentially connected subjects is possible. However, the calculation time becomes problematic if there are too many sequentially connected subjects or if the individual subjects have a high complexity. With the current implementation 3 or 4 subjects seems to be the maximal for this scenario. The number of subjects could be improved if the implementation relied more on memory use. The "always best option" calculation remains a problem for this kind of scenario. Because the calculation time is already reasonably high, it is difficult to decrease memory use. From this, we conclude, that it is important to allow the scenario author to disable "always best option" calculations from the rapport.

7.3 A scenario with parallel subjects

In our analysis of the number of possible paths, we noted that in general, subjects connected in parallel have the most significant effect on the number of paths. We also noted that interleave points almost have the same effect as an extra subject connected in parallel but require less work from the scenario author. Scenario 9 has multiple subjects connected in parallel and contains interleave points in each of those subjects.

The individual subjects of this test scenario are a lot less complex than those of the first scenario. The highest node count in a single subject is 47. The lowest node count in a single subject is 25. The lowest node count is 2. The nodes themselves are mostly sequentially connected. With 2, sometimes 3,

nodes connected in an exclusive choice. The scenario contains 4 subjects that are connected in parallel to each other. The scenario contains a total of 6 interleave points.

Name	Number of subjects connected in parallel	Number of interleave points	Calculation Time (ms)	Calculation Time (minutes)	Memory Use
9A	1	0	193	0.0032	21.2 MB
9B	2	0	19,728	0.3288	21,1 MB
9C	2	1	26,502	0.4417	179.1 MB
9D	2	2	27,659	0.46	234.1 MB
9E	3	2	963,791	16.06	2.4 GB
9F	3	3	OUT OF MEMORY		

Table 7.6: Best Path Calculation

Name	Number of subjects connected in parallel	Number of interleave points	Calculation Time (ms)	Calculation Time (minutes)	Memory Use
9A	1	0	3,013	0.050	156 MB
9B	2	0	OUT OF MEMORY		

Table 7.7: Complete Scenario Smell Analysis Calculation

With this scenario, memory is the limiting factor. If we have only two subjects connected in parallel both memory use and calculation time easily stay within acceptable bounds. There is a more substantial calculation time jump when adding a second subject, and a more substantial memory use jump when we add our first interleave point. Both calculation time and memory use start to rise significantly when we add a third subject. However, with 16 minutes the calculation time is still quite acceptable. Meanwhile, the 2,4 GB memory starts to become a problem. Once we add a third interleave point the program reaches the memory limit of 4 GB.

As with calculating the best path, for the complete report memory also is the limiting factor. The earlier discussed factors of a high memory footprint for

interleave points and the "always best option" calculation, play a part in this problem. Another reason might be the parameter P4. Both in scenario 2B and 2C, the OUT OF MEMORY occurs while calculating the greedy-search tree for P4. Memory use of a greedy-search tree depends on the number of parameter increases. At every node, the program remembers all the child nodes with an equally optimal parameter increase. If no child node increases the parameter(s) that are used for evaluating the paths, all nodes are placed on the stack of the search algorithm. This means that some parameters are more likely to result in memory problems than other parameters.

From this scenario, we can conclude that memory use is a problem. Both interleave points, and infrequently used parameters contribute to this problem. In this specific scenario, the calculation time is very low. It might, therefore, be acceptable if the computer is unusable while running the program, which would drastically increase the memory limit. The individual parameters also seem to be a more significant problem than the total score. Providing the users with an option to only calculate the total score, might alleviate the memory problems.

7.4 A scenario with high subject complexity

Scenario 10 has 4 subjects. The complexity of these subjects is higher than the subject complexity of the other test scenarios. The node count of the subjects is 92, 65, 78 and 31. Three of the subjects have a higher node count than any other subject in our test cases. Most nodes give the player a choice between three different nodes. However, there are a few nodes that give the player as many as eight choices.

Name	Number of subjects	Total number of nodes	Calculation Time (ms)	Calculation Time (minutes)	Memory Use
10A	1	92	45,574,251	759.57	26 MB
10B	2	157	OUT OF TIME		

Table 7.8: Best Path Calculation

In our theoretical analysis, we noted that a high node count and a large

Name	Number of subjects	Total number of nodes	Calculation Time (ms)	Calculation Time (minutes)	Memory Use
10A	1	92	OUT OF TIME		

Table 7.9: Complete Scenario Smell Analysis Calculation

number of parallel connections leads to a large number of paths. A high path count, will not necessarily impact memory use, but will always impact calculation time. The results of this test scenario support this thesis. Even with one subject, the calculation time is already higher than most interviewees are willing to wait. Meanwhile, with only 26 MB used, memory use is not a problem.

We can only conclude that supporting scenarios with a high subject complexity is very hard. Complexity is somewhat dependent on the number of nodes within one subject. We recommend that the program preprocesses the scenario, before it performs the optimal path calculations. If one of the node counts is higher than a specified number, the user should receive a warning about the possibly considerably large calculation time. Our current test data would suggest a maximum node count of 60 if a couple of hours is acceptable, and a node count of 50 if the calculation time should stay within one hour.

Chapter 8

Discussion

8.1 Answer Research Question

At the start of our research, we described the current state of dialogue training software and Communicate specifically. We noted that many dialogue training software use structured dialogue scenarios, written by experts in the area of communication skills. With the rapid advancement of computing technology, interactions between computers and humans are becoming more realistic. The decreasing reliance on new technologies means that individual scenario authors have a more significant influence on the effectiveness of the software.

We stated the following research question: Can we provide scenario authors with the information they deem important for the improvement of scenario quality? We formulated three smaller research questions to support this bigger research question:

- Is scoring one of the more difficult parts of creating a scenario?
- What information might help with improving the scoring?
- What information can be calculated, both in theory and in practice?

We will now discuss the results for each of these questions, as well as the overall research question.

By interviewing scenario authors, we obtained information about the current possibilities and difficulties in creating Communicate scenarios. We concluded that scoring is an important, and sometimes difficult part, in scenario creation. Most scenario authors use Communicate as a learning tool and not for an actual assessment. A 100% correct scoring is, therefore, not required. However, the scenario authors also stated that the scoring is essential for the learning effect of a scenario and that it affects the player's feeling of fairness.

Scenario writers consider the optimal score to be one of the most important aspects of a scenario. This is mostly because of the effect on the player's feeling of fairness. Currently, scenario writers often calculate the optimal score by hand, to make sure it is correct. Other scoring effects, like the path with the lowest score, or the longest path, are considered less critical. There are also some scoring effects, like subject monotone and correlation, that are difficult to calculate by hand. Considering these factors, we made the optimal score our primary focus for this research.

To determine what information might help authors with improving the scoring, we interviewed scenario authors about their inherent beliefs about correct scoring in scenarios. While some common trends are visible in these beliefs, there also is much variation. We compiled these beliefs into a list of scenario smells. Scenario smells are scenario qualities that we define as an indication of a possible faulty design. We based this approach on the use of code-smells in software development. Scenario smells we found are:

- Always choosing the optimal option does not result in the maximum score
- A longest path does yield the maximum (or minimum) score
- A shortest path does yield the maximum(or minimum) score
- The maximum and minimum score are unobtainable
- Two parameter have a high correlation

- The sign of the correlation of two parameters does not match expectations
- A scenario is not subject monotone

In investigating what information could be calculated, we focussed on the optimal path. We concluded that a search algorithm using a heuristic is not possible for Communicate scenarios, because there does not exist a reliable prediction method of future parameter changes on a path. We need to use a pathfinding algorithm, that uses brute-force to evaluate all the different paths. We have shown, both in theory and in practice, that a brute force method is feasible for most scenarios. For very complex scenarios, using brute force leads to significant increases of calculation time and memory use. While not impossible, using a brute force method in these cases seems inadvisable.

We looked at specific scenario aspects and how these influence calculation time and memory use if a brute force method is used. We concluded that parallel connections and interleave points have the biggest influence on calculation time. In practice, sequentially connected subjects are the biggest problem, because this type of connection is more common in actual scenarios. Parallel connections are the biggest factor in memory use. Another important aspect for memory use is the frequency with which a parameter is changed by individual nodes.

We created a common language that can be used to talk about scenario scoring and potential scoring pitfalls. We also created a program that creates a report about a scenario, which scenario authors can use to improve the scoring of a scenario. We conducted follow-up interviews, which show that scenario authors appreciate our method, and believe that it would improve the quality of scenarios. Limitations in hardware need to be considered and might exclude us from using this method for specific complex scenarios. Overall we conclude that for most scenarios it is possible to provide scenario authors with better information about their scenario.

8.2 Recommendations

Overall we recommend the creators of dialogue training software to consider implementing scenario smells into their tools. Our investigations have shown it to be a good tool for assisting dialogue writers with creating a scenario. The exact implementation can vary depending on the software and the user requirements. We do have some general recommendations and considerations.

We recommend the use of a brute force approach in calculating and evaluating paths. The brute force approach is usable for every dialogue training software that uses DAG's. There are also no special conditions that need to be placed on scenario design. Therefore, the brute force approach has the greatest degree of freedom for both the programmer and the scenario writer. To implement a brute force approach we also do not have to make any changes to the existing software

The brute force approach has two big drawbacks. Firstly, it requires the scenario writer to wait while the scenario is being processed. Secondly, the approach can be (almost) unusable for more complex scenarios. However, our investigations have shown that for most scenarios these drawbacks are within acceptable limits. We, therefore, conclude that this approach is still preferable to other methods.

There are some tradeoffs that might be considered to limit the drawbacks of the brute force approach. Creators of dialogue training software might consider limiting the scenario smell research to single subjects only. Our initial interviews have shown that while considering the entire scenario is preferable, limiting the investigation to single subjects is acceptable. In our tests, we found no single subject for which the scenario smell analysis could not be completed within a reasonable amount of time. In cases where it is important that the scenario smell analysis would never fail to complete, single subject analysis is preferable.

Another tradeoff that needs to be considered, is the tradeoff between calculation time and memory use. In general, we tried to limit memory use. The only exception being the greedy search tree, where we save every node that

still needs to be explored. Any new implementation of a scenario smell analysis should reconsider the tradeoff between these two factors. The choice will largely depend on if the software is run on a server or on a local machine of the scenario author. If the analysis is completed on a personal computer, we would recommend a bigger memory use and limiting the calculation time. For most servers the memory use is already reasonably high, so we would recommend our current implementation.

For implementations where memory use is a concern, we would also recommend not to use a greedy search tree. Our investigation has shown this search tree to have an overall worse performance than the depth-first tree. This is especially true for memory use. Removing the greedy search tree would also mean excluding the scenario smell "Always choosing the best option does not result in the maximum score". This scenario smell is rated pretty high by scenario writers. This means that excluding the scenario smell is far from preferable, but needs to be considered if the memory performance is an issue.

Overall we have concluded that it is acceptable if a scenario smell analysis does not complete for every scenario. In practice, this would mean that the analysis needs to be broken off after a certain time. Some scenario elements cause exponential growth of the calculation time. This means that the calculation time could potentially be days or weeks. We need an artificial time limitation to ensure that the system does not become unavailable for an unacceptable large amount of time. We used a limit of 24 hours. A limit of 7 hours would probably also suffice. In our investigation none of the analyses that ran longer than 7 hours, finished within the larger limit of 24 hours.

We also recommend the scenario writer receives a warning if calculation time is expected to be longer. The existence of a subject with more than 50 nodes or two or more subjects connected in parallel are good indications of long calculation time. Both can be detected in a preprocessing phase.

Lastly, we recommend that running a complete scenario smell analysis is not the only option. The scenario writer should be able to perform an analysis of a single smell or subject. The complete scenario smell analysis is the most elegant solution for the problems with scenario writing. However, in most cases, scenario writers will not use or need this complete tool, while the performance is severely limited by performing a complete analysis. It is,

therefore, preferable, if smaller analyses are supported. In order to keep the program simple and understandable, we would recommend that the complete analysis remains the primary option and that smaller analyses are a back-up or advanced option.

Chapter 9

Future Research

9.1 Workflow of Scenario Writers

In the recommendations chapter, we already discussed the trade-offs that we need to make between calculation time, memory use and completion rate. A better understanding of the influence of these three factors on the usability of the analysis and the overall quality of scenarios can help us improve on dialogue training software. There is a broad scope of possible further research, but two approaches stand out for us as having the most potential.

Firstly, we would propose a user study focused on scenario writers who have experience with using scenario smells. In our research, the user studies were limited to interviews with scenario authors about potential or relatively new tools. A user study focused on scenario authors with some experience with scenario smell analysis can provide us with more information about the relative importance of calculation time, memory use and completion rate.

A detailed analysis of the workflow of scenario authors could also make a further distinction between practical and theoretical considerations. Our current approach very much focused on how scenario authors view their process, not on how these processes actually work. It could very well be that there exist meaningful differences between these two.

Secondly, we would propose an extensive analysis of real-life scenarios. Our research focused on mathematically analyzing how different scenario elements impact the calculations. We then tried this out on a few actual scenarios. Repeating this experiment on a larger scale, with significantly more scenarios, could help us better understand the actual impact of scenario elements. If the limits of scenario smell analysis can be better defined, a better warning system could be developed.

9.2 Pruning

We have described why most traditional path-finding algorithms will not work for Communicate scenarios. Because of the lack of a viable heuristic and the requirement that the algorithm is 100% correct, we used a brute-force algorithm. The limiting factor of this approach is the size of the search space. If the search algorithm were to be improved, we need a way to limit this search space.

One possible way to achieve this is by using pruning. With pruning we disregard partly explored paths that are redundant, because their result is equal to that of another already explored path. If the 100% correctness requirement is relaxed even paths that are unlikely to result in an optimal path can be removed. This approach would decrease the search space and, therefore, the required calculation time. There would, however, be an increase of calculation time because of overhead. The memory use would also increase because more paths need to be remembered.

The actual effects of such pruning would need to be investigated. As stated pruning does have some negative effects on memory use and calculation time. For pruning to still be an effective measure, the number of pruned paths needs to be large enough. A study of current Communicate scenarios might determine how big a pruning can be performed. Alternatively, a new scenario smell analysis tool could be developed that uses pruning. We could then compare how the two tools perform on similar scenarios.

We expect that the most successful pruning method would be pruning between sequentially connected subjects. There is a clear division between

nodes that precede and succeed such a pruning point. We would propose that the algorithm prunes every path with the exact same parameter scores. This would guarantee the correctness of the algorithm, which is an important requirement. A drawback of this method is that it becomes less effective if there are a lot of parameters. An advantage of this method is that it is most effective if parameters receive little change. Our current algorithm performs very badly if parameters are rarely changed, so this type of pruning could potentially solve this issue.

9.3 Learning Effect

Our research has focused on the needs of scenario authors. How can we help scenario authors better perform their job, without limiting their freedom in scenario design? The underlying assumption is that scenario authors are experts in their fields and therefore the best source of information. In practice, this might not always be the case. Not all scenario authors are equally knowledgeable about either their field of practice or communication skills education. Further research might investigate if the use of scenario smells can not only help scenario authors but also improve the actual quality of scenarios.

One approach for investigating the quality of scenarios and the use of scenario smells is by looking at the learning effect, as popularized by Hattie [13]. A potential study might ask a scenario author to create a new scenario for a specific function. A group of students will then use this scenario as a training method. Afterward, they will partake in an assessment. We then introduce the scenario author with our tool and the defined scenario smells. The scenario author will improve on the scenario. A new group of students will use the improved scenario as a training method. They will also make an assessment. Afterward, we compare the assessments of the two groups to calculate the learning effect size.

Another approach that could be taken is comparing different playthroughs. This experiment would also need a new scenario, which the scenario authors then improves on with the help of our tool. The question is if there is a noticeable difference in play style between players who play the original

scenario and those who play the improved scenario. A possible measurement of these playthroughs might be Cronbach's alpha. We would expect that the consistency between the playthroughs would increase for the improved scenario because the players would take the higher scoring paths more often.

These two approaches can also help scenario authors get a better understanding of what design principles are effective in dialogue training scenarios. In programming, code-smells are often used in combination with software design patterns. The development of scenario design patterns alongside scenario smells will give scenario authors even more tools for creating effective dialogue training scenarios.

Chapter 10

Conclusions

In this thesis, we described the current state of Communication Education Training. We hypothesized that assigning accurate scoring parameters to Communicate is difficult, time-consuming and prone to errors. Our research has shown that this is indeed the case. Scenario authors describe working with parameters as time-consuming. The scenario authors also stated that they often avoid the more complex features of Communicate. Two common reasons for this avoidance is the lack of perceived worth compared to the effort and the fear of making faults.

To remedy the problems with scoring in Communicate scenarios, we proposed the concept of Scenario Smells. Scenario smells are properties of scenarios that indicate a possible fault in the scoring of a scenario. An essential aspect of scenario smells is that an editor can validate them automatically, lessening the workload of scenario authors. Based on interviews with scenario authors we created a first set of scenario smells. Further interviews have shown that scenario authors see these scenario smells as useful. Scenario authors also believe that the scenario smells can improve the quality of scenarios.

We created a sample program that can evaluate the existence of possible scenario smells. Our research has shown that, for most scenarios, this program can generate a scenario smell rapport within a timeframe that is acceptable for scenario authors. However, our research has also shown that for more complex scenarios the calculation time and memory use increase at a rapid

rate. In these cases, the use of scenario smells is impracticable but possible.

We recommend that the creators of Communicate, and other dialogue training software, implement our proposal of Scenario Smells. For most scenarios, the use of scenario smells will have a positive effect on the working experience of the scenario author and the usefulness of the scenario. We also recommend that the evaluation program informes the scenario author of possible problems with complex scenarios, by generating warnings based on the properties of a scenario.

Bibliography

[1] Marianne Berkhof, H. Jolanda van Rijssen, Antonius J.M. Schellart, Johannes R. Anema, and Allard J. van der Beek. *Ective training strategies for teaching communication skills to physicians: An overview of systematic reviews*. Patient Education and Counseling, 84(2):152–162, 2011.

[2] Tibor Bosse and Simon Provoost. *Integrating conversation trees and cognitive models within an eca for aggression de-escalation training*. In Qingliang Chen, Paolo Torroni, Serena Villata, Jane Hsu, and Andrea Omicini, editors, Proceedings PRIMA 2015: the 18th International Conference on Principles and Practice of Multi-Agent Systems, volume 9387 of LNCS, pages 650–659, 2015.

[3] Ana Paula Cláudio, Maria Beatriz Carmo, Vítor Pinto, Afonso Cavaco and Mara Pereira Guerreiro. *Virtual Humans for Training and Assessment of Self-medication Consultation Skills in Pharmacy Students*. In Proceedings ICCSE 2015: the 10th International Conference on Computer Science Education, pages 175–180, 2015.

[4] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[5] Patrick Gebhard, Gregor Mehlmann, and Michael Kipp. *Visual scenemaker—a tool for authoring interactive virtual characters*. Journal on Multimodal User Interfaces, 6(1):3–11, 2011.

[6] Johan Jeuring, Frans Grosfeld, Bastiaan Heeren, Michiel Hulsbergen, Richta IJntema, Vincent Jonker, Nicole Mastenbroek, Maarten van der

Smagt, Frank Wijmans, Majanne Wolters, and Henk van Zeijts. *Communicate! — a serious game for communication skills.* In Proceedings EC-TEL 2015 Design for Teaching and Learning in a Networked World: 10th European Conference on Technology Enhanced Learning, volume 9307 of LNCS, pages 513–517. Springer International Publishing, 2015.

[7] Raja Lala, Johan Jeuring, Jordy van Dortmont, and Marcell van Geest. *Scenarios in virtual learning environments for one-to-one communication skills training.*

[8] Anton Leuski and David Traum. *NPCEditor: Creating Virtual Human Dialogue Using Information Retrieval Techniques.* AI Magazine from the Association for the Advancement of Articial Intelligence, 32(2):42–56, 2011.

[9] Jonathan Posner, James A. Russell, and Bradley S. Peterson. *The circumplex model of affect: An integrative approach to affective neuroscience, cognitive development, and psychopathology.* Development and Psychopathology, 17(3), pp. 715–734, 1980.

[10] Jeroen Wauters, Frederik Broeckhoven, Maarten Overveldt, Koen Eneman, Frederik Vaassen, and Walter Daelemans. *delearyous: An interactive application for interpersonal communication training.* In Serious Games: The Challenge: Joint Conference of the Interdisciplinary Research Group on Technology, Education, and Communication, and the Scientic Network on Critical and Flexible Thinking Ghent, volume 280 of Communications in Computer and Information Science, pages 87–90. Springer, 2012.

[11] Edsger W. Dijkstra. *A note on two problems in connexion with graphs.* Numerische mathematik 1(1), pages 269-271, 1959.

[12] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. *A formal basis for the heuristic determination of minimum cost paths.* IEEE transactions on Systems Science and Cybernetics 4, no. 2, pages 100-107, 1968.

[13] Hattie, John. *Visible learning for teachers: Maximizing impact on learning.* Routledge, 2012.

First Interview

General Information

1. What is your profession, what is your working domain?
2. How many scenarios have you created with Communicate?
3. Have you used Communicate in a class room situation?
4. Do you make use of the validate option?
5. Do you make use of the see parents option?
6. Do you make use of the calculate points option?
7. Do you usually work node for node (completing, the dialogue, prerequisites, emotions, parameters) or do you first write all the dialogue and then at parameters and prerequisites?

Their Ideas

8. What kind of problems do you experience in creating Communicate scenario's?
9. Are there problems that are specific to scoring?
10. Do you have any ideas how we could solve those problems?
11. What kind of steps do you take to avoid problems in scoring?

Assumptions

I now would like to ask you some questions about scenario's. The editor gives you a lot of freedom in how scenarios should work. This can be a problem for creating validation tools. We can try to make some assumptions about scenario's however that can help us define what is a 'good' scenario. Can you describe how true you think the following assumptions are. You have the following options: always true, mostly true, can't say, mostly false, always false.

12. The longest path yields the highest score.
13. If the player chooses the best option (the option with the highest score increase) at every node, he / she should have the highest possible score at the end of the scenario.
14. The sequence in which subjects have been dealt with, should not mater for the end score.
15. It should be possible to get the maximal score.
16. It should be possible to get the maximal score on every parameter, but not necessarily the maximal end score.
17. Every node (even those with prerequisites) should be reachable in some way.
18. Parameters should have no correlations with each other.
19. The longest path should yield the lowest score
20. It should be possible to get the minimal score
21. It should be possible to get the minimal score on every parameter, but not necessarily the minimal end score.
22. The player should sometimes sacrifice a scoring opportunity to receive a better opportunity down the line.
23. Switching from subject is never a good idea.

24. All the (scoring) parameters should use the same scale (1 – 10, 0 – 100 etcetera).
25. It should be possible to score above 50
26. Parameters are always positive.

Can you tell me the three assumptions that you currently find the hardest to validate in your own scenario's? Are there any assumptions we have missed?

Optimal Path

In discussions about (dialog) trees we often talk about optimal paths. This is a term that usually described the sequence of nodes that rewards us with the most wins (in games like chess) or the highest score. We could therefore conclude that the path with the highest overall score is the optimal path in the case of Communicate, but this is not necessarily satisfactory. I will now state a couple of ways in which we could determine the optimal path. Can you tell me for each definition if you agree or disagree that this would be a workable definition?

27. The highest average of all the parameters (no weights)
28. The path that includes the highest score for an individual parameter
29. The weighted highest average of all the current parameters (current total score)
30. The quickest path
31. The path that has the highest score on the lowest parameter.
32. The slowest path
33. Same as one of the above, but with the extra requirement that all the scores need to be above a certain threshold

Which of these definitions do you thinks works the best? Are there any other possible definitions that we didn't think of?

Solutions / UI

I know want to talk about possible improvements of Communicate. These are hypothetical improvements, so we don't need to think about feasibility. For each of this improvements, can you tell me what you think about the following aspects:

- Will using it improve your scenario's?
- Do you think you would make use of this improvement?
- How much margin for error is there, ea. the tools needs to give me an exact answer, the tool needs to give me a reasonable answer etcetera.
- Would it be okay if we ignore some part (requirements, subjects etcetera) of the scenario in the tool?

You don't need to give me an exact answer, so feel free to give me all your thoughts.

34. Visually displaying all the parameter changes for every node.
35. Color coding all the nodes based on the parameter changes. Green for a positive effect on the overall score, orange for no effect and red for a negative effect.
36. Visually displaying the change in the overall score for each node
37. A button which shows the optimal path through a subject
38. A button which shows the optimal path through the entire tree
39. A button which shows the optimal path from a specified node
40. An option to repeatedly show optimal paths from different start nodes, whereby the start points are randomly chosen by the computer.
41. An option to mark nodes as desirable and to get a warning if a desirable node isn't included on the optimal path

42. An option to mark nodes as undesirable and to get a warning if a desirable node is included on the optimal path

43. Statistics about the scores like correlation, minimal value, maximal value etcetera

44. A profile of every subject that shows you how often a parameter is used in that subject, what the maximum changes (both positive and negative) are etcetera.

Finishing Thoughts

That was my last prepared question. Now that we have finished the entire interview, do you have any last thoughts. Did you think of any new possible improvements that could be made to Communicate in respect to scoring? Did is miss something important in the discussion about this subject?

Second Interview

General Information

1. How many scenarios have you created with Communicate?
2. Are you currently using Communicate in a classroom situation?
3. What is the context of your example scenario?

Demonstration

We will now demonstrate the pathfinder. Afterwards, we want to discuss the different scenario smells. We want to focus on the following aspects:

- Was the result similar to what you expected?
- If not, what was different?
- Would you make changes to this scenario after seeing the result?

We will discuss the scenario smells in this order:

4. Always choosing the best option does not result in the maximum score
5. A longest path does yield the maximal score
6. A shortest path does yield the maximal score

7. Is it possible to obtain a parameter value lower than the specified minimum value?
8. Is it possible to obtain a parameter value higher than the specified maximum value?
9. Is there any correlation between parameters?
10. Is the scenario subject monotone?

Our Program

The example program we are about to demonstrate has multiple features. We want to know how interested you are in each of them. Can you describe how likely it is you would use these functions by giving a number between 1 and 5, where a 1 means "would not use at all", and a 5 means "would definitely use"?

11. Calculating the best path (overall score)
12. Calculating the worst path (overall score)
13. Calculating the longest path
14. Calculating the shortest path
15. Checking if always choosing the best option will lead to the maximal score
16. Checking if the longest path will also be the best path
17. Checking if the shortest path will also be the best path
18. Checking if the minimal and maximal value of a parameter is attainable
19. Calculating the correlation between different parameters
20. Checking if a scenario is subject monotone

Conclusion

21. Do you think that the demonstrated program might help you improve your scenario's?
22. Are you likely to use the program if it remains separate from the Communicate website?
23. Are you more likely to use the program if it is integrated in the Communicate website?
24. Where the calculation times a problem for you?
25. For your personal use, what would be the most useful feature of the current program?
26. Are there any features that you would like us to expand on?