Master Thesis

Instructor Supporting Model

A Model for Supporting Instructors in using Serious Games

Author:
R. de Jong
Student ID: 3812065

Supervisors:
Dr. A. Heuvelink (TNO)
Prof. Dr. J-J.Ch. Meyer (University Utrecht)

Master's programme
Cognitive Artificial Intelligence (30 ECTS)

June 11, 2014

Contents

1 | Introduction

Serious gaming is an up and coming form of interactive education. Serious games provide a controlled environment in which virtual training can take place. Using serious games as a training environment provides a number of advantages over real world training: Serious games can be used to train people in a (virtual) scenario that would be dangerous if done in the real world. Virtual training in serious games is also less expensive and less time-intensive than training in real life, the only investment needed are computers and the serious gaming software. From then on out, creating and executing scenarios is easy to do. Without a virtual environment, the environment would have to be constructed and maintained. Creating an environment this way would be more costly and more time-intensive. Moreover, the environment would only be available for one group of trainees at a time, and they would have to travel towards the physical location, which results in travel costs. Finally, artificial intelligence can take over the roles of certain roleplayers in serious games, which results in needing less people to carry out a training session.

A serious game is used to teach the player, commonly referred to as trainee, something. The trainee usually has one or more tasks he has to complete during the serious game that relate to the learning goals of the trainee. During the course of the serious game, there is usually a secondary party involved. This party is called the instructor. The instructor has as goal to optimize the learning experience of the trainee. This can be done in a variety of ways: by directly influencing the game, by communicating with the trainee, and/or by giving feedback at the end of the training session.

1.1 Problem definition

The Dutch military has requested the Netherlands Organization for Applied Scientific Research (TNO) to research the military's arsenal of serious games in order to make it more accessible. This has spawned the "Vergroten Toegankelijkheid van Serious Games" project within TNO (V1313 VT-SG, project nr. 053.03307). Within this project, a multitude of serious games used within the Dutch military have been, or are being evaluated. One of these serious games is Virtual Battlespace 2 (VBS2), a serious game that acts as a battlefield simulation. The Dutch army mainly uses VBS2 for tactical training of their infantry, and during the evaluation of VBS2 at TNO several issues concerning accessibility became apparent. The focus of this thesis is investigating the ways in which these encountered issues could be handled to ensure that VBS2, and other serious games, will be more accessible to its users.

The research we did concentrated on improving the way the game supports the instructor in his or her actions. The instructor has a variety of tasks that need to be accomplished before, during, and after a training session. Before the training session, the instructor needs to know all the details surrounding the training session; the learning goals related to the training scenario and what will, or might happen during the training. During the training session, the instructor needs to ensure that the trainee learns as much possible, by adjusting the difficulty of the scenario to fit the trainee's experience, adding new events to the scenario on the fly, or by controlling objects within the scenario. Finally, after the trainee has completed the training scenario, the instructor should give feedback to the trainee based on his performance during the session.

VBS2 offers the instructor relatively little support when performing these tasks. The tools provided by the serious game are specialistic in nature because they were designed for someone who has expertise in creating training scenarios. However, the instructor is usually not the person who created the training scenario, as such, he has little expertise in the usage of these tools. Since the game is made to fit a multitude of training domains, there is no knowledge of the learning goals for the trainee inside the game, as these learning goals are specific to a domain. As a result, there is a disconnect between the purpose of the scenario and the scenario as it stands within the serious game.

Due to the specialistic tools within the game, it is difficult for the instructor to control the scenario by adjusting the difficulty, adding events, or controlling objects. Especially when the instructor has to multitask and do a multitude of these tasks at once.

Summarizing all of this, this thesis focuses on the question *How can we support the instructor in providing the optimal learning experience for the trainee?*

1.2 Focus

To support the instructor in the variety of tasks he has to accomplish, we have created a supporting model dubbed as the Instructor Supporting Model, also referred to as ISM. Our model contains a variety of user interface components and A.I. techniques that are used to provide ample support for the instructor during the key parts of the training session. Our model focuses on a balance between flexibility and usability, and a balance between user control and A.I. support.

The main focus of the Instructor Supporting Model is to provide support for the instructors within the Dutch army. More broadly however, we wish to create a model that can serve as a base for supporting instructors in any serious game, not just military scenarios. We have specifically designed our model in a way that extension is possible. However, we only test the model within the boundaries of the concept behind a military training scenario.

1.3 Outline

First, we discuss existing theory behind serious games and psychological concepts that are relevant to our project, and how they relate to the Instructor Supporting Model.

We then introduce the Instructor Supporting Model by analysing the current situation that holds for VBS2, providing solutions and optimizations, and then fleshing out these solutions and optimizations.

We also implemented the Instructor Supporting Model by creating a prototypical serious game and applying the ideas behind ISM within this serious game. This prototypical serious game is based on the virtual training scenarios used by the Dutch army. By selecting an appropriate game engine and game type, we are able to successfully create a serious game that adapts the ideas behind these virtual training scenarios.

The Instructor Supporting Model is then tested by having test subjects play the instructor during a training session in the prototypical game with, and without the model. Data is collected within the game and with the use of some external programs.

After, we analyse the data to test whether the Instructor Supporting Model successfully supports the instructor in creating an optimal learning experience. Followed by our discussion regarding the data we collected. Finally, we present our conclusions and recommendations for future research.

2 | Background

The instructor is an important addition to serious games, as it has been shown that minimal guidance does not work well when instructing a student (Kirschner et al., 2006). As such, it would be much more difficult for the player (student) to learn something when playing a serious game unguided by an instructor.

In this section we will start off by introducing Virtual Battlespace 2, the serious game used in the training relevant to our research. After that, we will go into a number of psychological concepts that we deem important for serious gaming. Specifically, we will look at how these psychological concepts can be influenced by the instructor.

2.1 Virtual Battlespace 2

Virtual Battlespace 2 (VBS2) is used by the Dutch military for tactical training of its infantry. VBS2 is developed by Bohemia Interactive Simulations (Interactive, a) and is the successor to Virtual Battlespace 1 (VBS1). It is specifically tailored towards military and first-responder-themed training environments. VBS2 makes use of the Real Virtuality 2 game engine.

VBS2 is a battlefield simulator in which the players (trainees) each control a single soldier from the first-person perspective. Trainees move around in a 3-Dimensional virtual world in which they have to carry out certain tasks (see figure 2.1). The combination of the design of the virtual world together with the set of tasks to be completed by the trainee is called a scenario, and a scenario is designed by a scenario maker prior to the training using the Offline Mission Editor (OME). During the game, trainees can encounter and operate a variety of military and non-military vehicles (land, air, and sea) as well as interact with structures (e.g. houses) and various other objects (e.g. bombs). VBS2 also provides Artificial Intelligence (A.I.) for computer-controlled objects and units within the game.

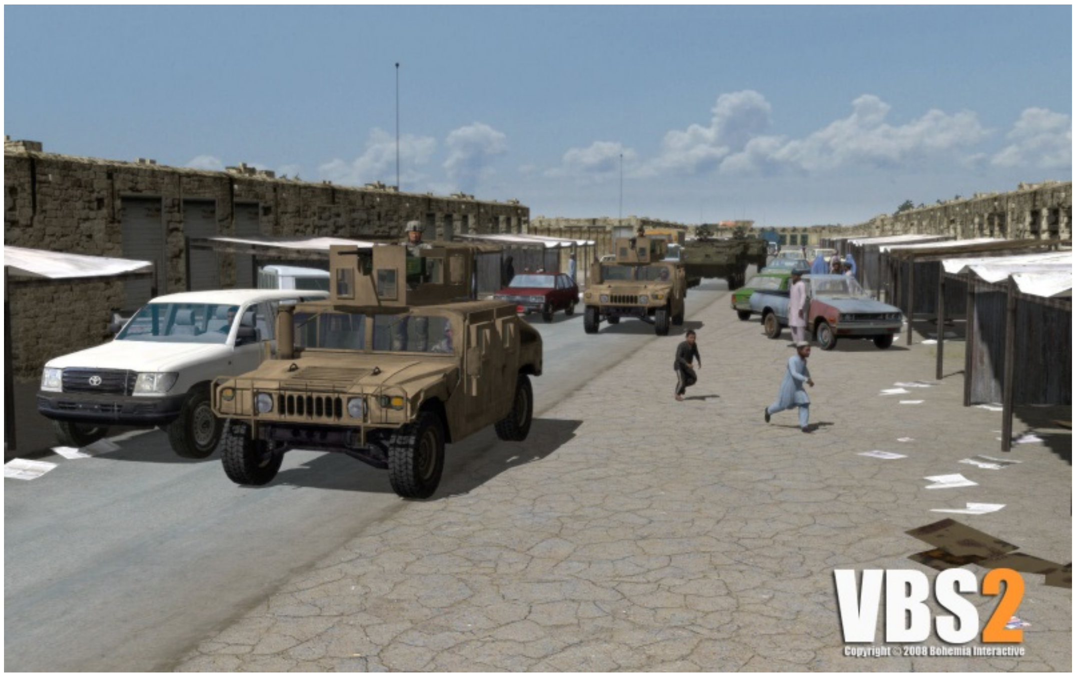


Figure 2.1: An example of VBS2 (not from trainee's perspective). Here you can see trainees in a convoy moving through a city.

In VBS2, trainees can interact in the game during group training sessions. VBS2 provides a squad-management system that allows trainees to issue orders to squad members (squad members can be player-, or computer-controlled).

During the scenario, VBS2 allows the instructor to alter the scenario in real-time using the Real-Time Editor (RTE). This editor allows the instructor nearly all the options that the OME provides with the same sort of user interface. It allows the instructor to add, alter, or remove objects and units. Additionally, it also allows the instructor to give non player-controlled (NPC) units orders (e.g. move to way point). Finally, RTE allows for a number of additional alterations, such as changing the weather.

After completing a scenario, VBS2 allows the opportunity to review the scenario using the After Action Review (AAR). The AAR is a replay system that allows the trainee(s), the instructor, and all who are involved to review the execution of the scenario in great detail. All actions of the trainee and all aspects of the virtual environment are recorded and can be freely seen as if on a video.

Developers can alter VBS2 to enhance the simulation experience. They can implement changes into the game by altering the game engine code or through the VBS2Fusion application programming interface (API). For example, VBS2 has been altered before to provide an accurate and realistic simulation of the FV4034 Challenger 2 battle tank (Interactive, b).

2.2 Flow

One important aspect in any kind of learning environment, including serious gaming, is the concept of Flow. This psychological concept was introduced by Mihaly Csikszentmihalyi (Csikszentmihalyi, 1990) and represents a positive phenomenon in which a person is completely immersed in their task. While being immersed in their task, the person is able to maximize his performance and therefore learn more. The Flow is defined by eight core elements. When a person is in the Flow, that person will experience at least one, but most of the time all of these elements (Csikszentmihalyi, 1990):

1. A challenging task that requires skill and that can be completed by the participant.
2. The ability to concentrate on the task.
3. Task has clear goals.
4. Clear and immediate feedback.
5. Acting with deep, but effortless involvement, that removes awareness of the worries and frustrations of everyday life.
6. Sense of control over one's actions.
7. Loss of concern for self during the Flow experience, and a stronger sense of self after the Flow experience.
8. Altered sense of duration of time.

In a serious gaming environment, a sense of Flow correlates to a player's[1] immersion in the game. Flow has been linked to whether or not someone enjoys playing a computer game through the GameFlow model (Sweetser and Wyeth, 2005). Sweetser's model links the eight core elements of Flow to game criteria. Most of these criteria are directly related to the game design, and are not relevant to our research. There are some criteria that are relevant because they can be directly influenced by the instructor, and these we will now discuss:

A challenging task that requires skill and that can be completed by the participant:

- Challenges in games must match the players' skill levels.
- Games should provide different levels of challenge for different players.
- The level of challenge should increase as the player progresses through the game and increases their skill level.
- Games should provide new challenges at an appropriate pace.
- Games should increase the players' skills at an appropriate pace as they progress through the game.
- Players should be rewarded appropriately for their effort and skill development.

All of these criteria can be directly influenced by the instructor in a serious game. The instructor should be familiar with the skill level of the player before starting a serious game. As such, the instructor can pick an appropriate scenario that fits the skill level of the player. Additionally, the instructor should have the tools available to change the challenge levels by adding, altering, or removing challenges during the game. Players can be rewarded by the instructor for example by the removal of a more tedious task.

The ability to concentrate on the task:

- Games should provide a lot of stimuli from different sources.
- Games must provide stimuli that are worth attending to.
- Games should quickly grab the players' attention and maintain their focus throughout the game.

[1]The player in a serious game is often called a trainee, we use player instead of trainee in this section because it correlates with the used literature.

- Players shouldn't be burdened with tasks that don't feel important.
- Games should have a high workload, while still being appropriate for the players' perceptual, cognitive, and memory limits.
- Players should not be distracted from tasks that they want or need to concentrate on.

All of these criteria can be directly influenced by the instructor in a serious game. The instructor usually adds events to the serious game as needed. As such, the instructor can add or alter events to provide stimuli, grab the attention of the player, or heighten the workload. The instructor could also remove events related to unimportant tasks, remove events related to distracting tasks, or lowering the workload.

Task has clear goals:

- Overriding goals should be clear and presented early.
- Intermediate goals should be clear and presented at appropriate times.

These criteria can be directly influenced by the instructor. Before having the trainee play the scenario, the instructor could clearly explain the goals of the scenario to the trainee. Furthermore, by adding or removing appropriate events, the instructor can reinforce certain goals, and even introduce new goals via obstacles.

Clear and immediate feedback:

- Players should receive feedback on progress toward their goals.
- Players should receive immediate feedback on their actions.

These criteria can be partially influenced by the instructor. If the player has finished a task relevant to one of the learning goals, the instructor can reward the player with a positive event in the game (e.g. giving the player an item in the game). Furthermore, the instructor could take control of the in-game objects to ensure more genuine feedback (e.g. if the player approaches a computer-controlled character in a correct way, the instructor could have that character react positively, and conversely, if the player approaches the character in an incorrect way, the instructor could have that character react negatively).

It is clear that the instructor can have a direct impact on the Flow of the player in the serious game, provided the appropriate tools are available. As the Flow is different per player (see figure 2.2), the instructor is a very necessary component in serious games to ensure that every type player can have a Flow experience as the required challenge level may differ between players. In addition, learning goals may also differ between players.

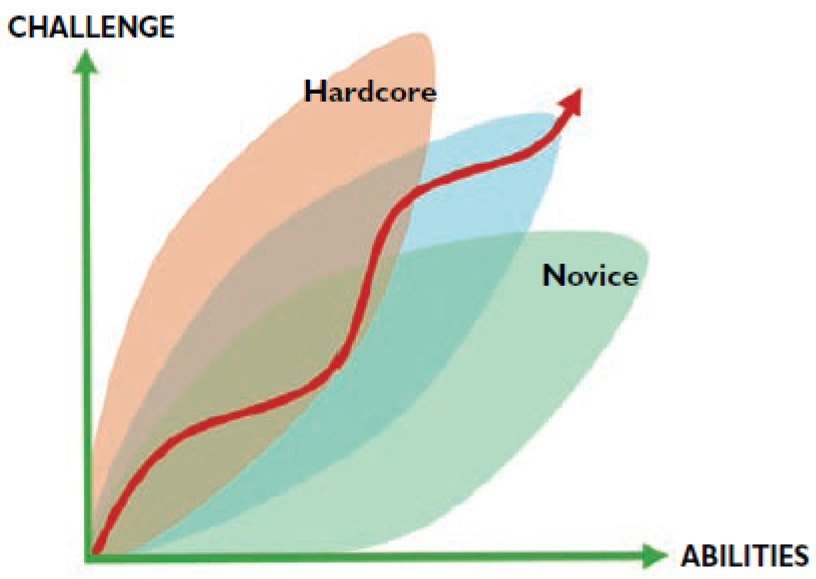


Figure 2.2: Different players have different Flow zones (Chen, 2007).

2.3 Intrinsic motivation

Thomas W. Malone has argued that there are four essential characteristics that make good computer games and other intrinsically enjoyable situations: challenge, curiosity, control, and fantasy (Malone and Lepper, 1987). Malone first introduced the three aspects of challenge, curiosity, and fantasy (Malone, 1980) and fleshed them out into a framework for a Theory of Intrinsically Motivating Instruction (Malone, 1981). Later he expanded this theory into a Taxonomy of Intrinsic Motivations for Learning and added the control characteristic (Malone and Lepper, 1987). Malone's taxonomy consists of the following elements:

1. **Challenge**
    - Goals.
      There should be goals for the trainee to accomplish.
        - Personally meaningful goals.
          These goals should be personally meaningful to the trainee.
        - Obvious or easily generated goals.
          The goals should either be obvious to the trainee, or they should be emergent, in which case it should be easy for the trainee to generate the goals for himself.
        - *Performance feedback.*
          The activities for the goals should provide feedback based on the trainee's performance. The feedback should be:
            * *Frequent.*
            * *Clear.*
            * *Constructive.*
            * *Encouraging.*

- Uncertain outcome.
  The outcomes of the environment should be (partially) uncertain to increase the challenge level.
  - Variable *difficulty level*.
    The difficulty level should be adjusted according to the trainee.
    - Determined automatically.
      The environment might be able to automatically adjust the level of difficulty for certain activities based on the trainee's performance.
    - Chosen by learner.
      The trainee can choose the level of difficulty.
    - Determined by opponent's skill.
      If there is an opponent in the environment, the opponent's level of skill would decide the difficulty level.
  - *Multiple level of goals*.
    Having multiple sets of goals can increase the challenge level related to these goals.
    - Score-keeping.
      Having goals related to score can invoke multiple goals (e.g. goal a: reach a score of 100, goal b: reach a score of 200).
    - Speeded responses.
      Having goals related to time can invoke multiple goals (e.g. reach goal a faster than you did previously).
  - *Hidden information*.
    Incomplete or hidden information provides uncertain outcome.
  - *Randomness*.
    Random elements also provide uncertain outcome.
- Toys vs. tools.
  Toys are objects without external goals, whereas tools are objects used to achieve external goals. Toys should be challenging, whereas tools should be efficient and not challenging.
- Self-esteem.
  Trainees can succeed or fail. Both have impact on their self-esteem. The trainee's self-esteem should not drop, because it will negatively affect the learning experience. In order to ensure maximum motivation, use personally meaningful goals.

2. **Curiosity**
   - Sensory curiosity.
     The use of sensory stimuli: audio and visual effects.
   - Cognitive curiosity.
     Prospect of modifying higher-level cognitive structures.
     - "Good form" in knowledge structures.
       Environments can stimulate curiosity by having the trainee believe that his existing knowledge structures are not well-formed. These are the characteristics of well-formed cognitive structures:
       - Complete.

        * Consistent.
        * Parsimonious.

3. **Control**
    - Contingency.
      The trainee may fail to a certain degree at certain activities by interacting in (partially) incorrect ways. It should be ensured that these interactions still have some effect on the activities, and that the trainee can succeed to a certain degree.
    - Choice.
      The environment should have some room for the trainee to make choices. Such as choosing which task to take.
    - Power.
      The environment's outcome should be dependent on the trainee's responses. Changing the environment based on the trainee's actions will give the trainee a greater sense of control.
4. **Fantasy**
    - Endogenous and Exogenous fantasies.
      In an endogenous fantasy, the fantasy the fantasy depends on the skill, and the skill depends on the fantasy. In an exogenous fantasy, the fantasy depends on the skill being learned, but the skill does not depend on the fantasy.
    - Emotional aspects of fantasies.
      Different fantasies suit different types of trainee. Every person has their own emotional choice of what kind of fantasy they prefer.
    - Cognitive aspects of fantasies.
      Different types of fantasies offer different kind of cognitive aspects that might provide the trainee with a better understanding of new information.

Within this taxonomy, a number of aspects can be directly influenced by the instructor.

2.3.1 Challenge

Goal: The *performance feedback* of the goal could be influenced by the instructor in certain situations by altering the scenario in a way that gives the trainee the feeling that he is achieving his goal. The instructor has the ability to make feedback (in-game feedback) *frequent*, *clear*, *constructive*, and *encouraging*. Since it is hard to make the environment react to everything the trainee does due to A.I. limitations, this type of responsibility could partially be transferred to the instructor. The instructor can then use his knowledge of the trainee to further enhance the feedback given by the game.

Uncertain outcome: The instructor can influence the *difficulty level* at any point in time by removing, altering, or adding tasks. Furthermore, the instructor can also influence the *multiple levels of goals* by ensuring that the scenario fits the trainee's recorded performance (e.g. having the trainee perform a task quicker than he has performed it previously). *Hidden information* is another aspect that can be enhanced by the instructor, since the instructor could add hidden objects and reveal them at any point in time. Finally, *randomness* is also enhanced by the instructor, since the instructor can decide to alter the scenario at any point in time or take over elements and make them act in a more random manner.

Self-esteem: By ensuring that the trainee is playing at an appropriate level of difficulty, the instructor can ensure that the damage done to the self-esteem of the trainee through failure is reduced.

2.3.2 Curiosity

Sensory curiosity: By adding appropriate objects on the spot, the instructor can seek to enhance a trainee's curiosity. This is especially true when a trainee misses an object that had the property to spark the curiosity of the trainee, in which case the instructor could simply add another copy of the object within the trainee's visual area.

2.3.3 Control

Contingency: The instructor can ensure that the scenario is properly adjusted to the trainee's strengths and weaknesses, allowing the trainee a greater sense of control.

Choice: The instructor could ask the trainee some questions before the scenario starts and using the answers to personalize the scenario (e.g. a specific learning goal on which the trainee wishes to focus). Furthermore, while playing the scenario, the instructor can appropriately act to choices the trainee makes (e.g. ignoring a problem), which ensures that the trainee gets a more genuine experience, in which all of the choices he makes are responded to.

Power: The instructor can further empower the trainee's influence on the scenario because the instructor can appropriately handle unexpected actions. If a trainee would make such an unexpected action, the instructor can ensure that the scenario would change accordingly (e.g. if the trainee would open fire inside a village, the instructor could make all the inhabitants flee).

2.3.4 Fantasy

Emotional aspects of fantasies: The emotional aspects of fantasies can be enhanced by the instructor through the same procedure as described at **Choice**. By having the instructor and the trainee discuss before the scenario, the instructor could choose and adapt the scenario in such a way that it fits the trainee's fantasy of choice.

2.4 Game difficulty

The difficulty of a task, or challenge, not only influences the trainee's Flow, but also directly affects the trainee's learning experience. We already covered the challenge characteristic of Malone's paper (Malone and Lepper, 1987).

First of all, the difficulty of a game, and the rate at which the difficulty changes, directly influences the player's (trainee's) immersion (Qin et al., 2010).

If the difficulty of the game is too high for the trainee, he will likely fail. Conversely, if the difficulty of the game is too low for the trainee, he will likely succeed. Jesper Juul's research shows that players (trainees) do not want to fail and that winning without failing leads to dissatisfaction (Juul, 2009).

Furthermore, if the game difficulty does not change over multiple sessions, the engagement of the player (trainee) can decrease (Chanel et al., 2008).

2.5 Conclusion

By applying the possibilities that a instructor brings to a serious game on the psychological concepts and frameworks discussed, we discover that there are a number of crucial psychological elements that an instructor can influence when supporting a trainee. Concluding from this, we say that the instructor has the following tasks to ensure that the trainee has an optimal learning experience:

- The instructor should ensure that the trainee does the tasks related to the scenario's learning goals. I.e. the trainee should not miss any tasks present in the scenario, nor should the instructor add tasks that are not relevant to the scenario's learning goals.
- The instructor should adapt the scenario to fit the trainee.
  - The instructor should adjust the difficulty of the scenario to fit the trainee's experience. I.e. tasks in which the trainee has a lot of experience should be more difficult, and conversely, tasks in which the trainee does not have a lot of experience should be easier.
  - The instructor should reinforce learning goals by adding additional tasks related to one or more learning goals if there is time and space in the scenario.
- The instructor should ensure that everything within the scenario responds according to the trainee's actions, and that these responses correspond to the trainee's performance.

Finally, we use Peeters' automated scenario-based training ontology as a base for our model (Peeters, 2014, chap. 6). Peeters' work focuses on personalized educational games, and automatizing these scenarios. Among other things, Peeters' discusses the use of (automated) scaffolding to adjust events to fit the trainee's competence level, and automated generation of scenarios. Peeters' ontology is very extensive and consists of three layers: the game world layer, the task domain layer, and the didactics layer. Not everything is relevant for us, however. we use the clearly defined relationships between the instructor, the trainee, the scenario, and the learning goals as inspiration for our own model (see figure 3.1) . Peeters was also involved in the "Vergroten Toegankelijkheid van Serious Games" group, and provided valuable feedback on our model.

3 | Model

In this chapter we will explain the details of the Instructor Supporting Model, also referred to as ISM. We want to be able to support the tasks we defined in the previous chapter. As such, we define the following requirements for a serious game that ISM should support per task:

- **Task:** *The instructor should ensure that the trainee does the tasks related to the scenario's learning goals.*
  The goals of the scenario should always be clear to the instructor. Furthermore, it should be clear what the tasks are in the scenario, and what task relates to what learning goal.
- **Task:** *The instructor should adapt the scenario to fit the trainee.*
  The instructor should be able to understand the strengths and weaknesses of the trainee. If the instructor does not know the strengths and weaknesses of the trainee before the scenario starts, he should be able to learn about the strengths and weaknesses during the scenario. Furthermore, it should be easy for the instructor to alter the difficulty of the scenario according to these strengths and weaknesses. Finally, it should be easy for the instructor to make changes during the scenario to reinforce certain learning goals according to the strengths and weaknesses of the trainee.
- **Task:** *The instructor should ensure that everything within the scenario responds according to the trainee's actions, and that these responses correspond to the trainee's performance.*
  It should be simple for the instructor to take direct control of objects within the scenario.

We will start off by elaborating on the current situation for VBS2 instructors in the Dutch army. We will highlight the aspects of the pre-session and in-session situations and present a model of the concepts behind a training scenario.

Secondly, we will explore the training session and list the problems and possible optimizations within the training session. These problems will also be linked to the solutions proposed and provided by ISM. These solutions and optimizations will then be elaborated upon at the end of the chapter.

3.1 Current situation

The *instructor* (see figure 3.1 for an overview of the entire situation with the appropriate concepts) wants to provide the optimal learning experience for the *trainee*. To achieve this, the instructor has to lead the trainee through a *scenario* within a serious game. This scenario focuses on one or more *learning goals*. The instructor wants to ensure that the trainee gains experience in these learning goals, with the eventual goal of the trainee reaching the learning goals.

Every scenario has a certain number of *events*. An event is an occurrence that happens within the game. It is either triggered by the instructor or it is something that is always there. The first is called a triggered event, the second is called a static event. An example of a triggered event would be "2 tanks drive from point A to point B", so when the instructor triggers the event, 2 tanks would appear at point A and drive to point B within the serious game. Every event relates to one or more learning goal. The instructor can therefore train the trainee in the learning goal by triggering these events. It is also possible for the instructor to add his own custom events during the game.

Furthermore, every event is expected to trigger a reaction from the trainee. This reaction can be roughly classified as *correct behaviour* or *faulty behaviour*. Correct behaviour means that the trainee correctly responded to the event, and is on the right track when it comes to the related learning goals. Faulty behaviour means that the trainee incorrectly responded to the event, and is on the wrong track when it comes to the related learning goals. Whether it is correct or faulty behaviour, reactions can lead to additional events called *responses*. These responses are specified as reactions to the trainee's action and can be activated by the instructor by the means of an event. This could be done for a variety of reasons, e.g. to reward the trainee, or to make it more difficult.

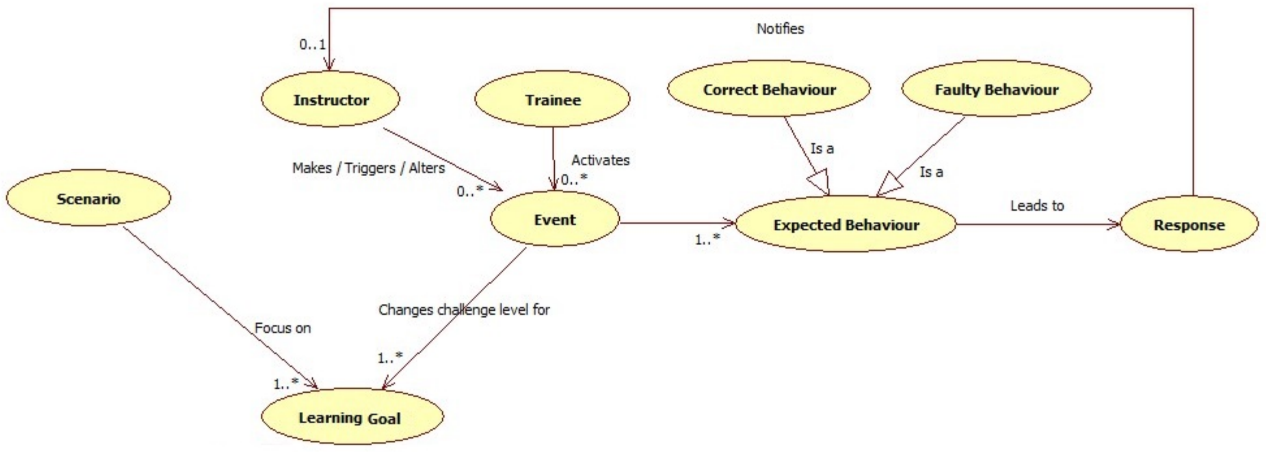


Figure 3.1: A model of the situation.

In order to provide the optimal learning experience, the instructor has a number of tasks for three stages of the training: pre-session, in-session, and post-session.

3.1.1 Pre-session

Before the training commences, the instructor needs to have an overview of the possibilities that are present within the scenario and, on a broader scale, within the serious game.

Specifically, the instructor needs to have a good overview of the in-game map used within the training. The in-game map refers to the in-game level in which the training will occur. The instructor also needs to have an understanding of all the events within the scenario and will also need to map these events to the map used in the scenario. Therefore, the instructor needs to know for every event where it takes place on the map and what happens when the event is triggered. Finally, the instructor also needs to know what kind of events he can create and control himself.

Currently, the scenario is presented to the instructor via an excel sheet, usually printed. The excel sheet describes the idea behind the scenario and the learning goals behind the scenario. It also contains a visual overview of the in-game map. Every event that can be triggered within the scenario is described in the excel sheet, and is indexed via a word or a short sentence (2-4 words). The events are textually described (i.e. when to trigger the event and what happens when the event is triggered) and their position is referenced via an image of the location in which the event will take place. This image can then be linked to the visual overview of the scenario map.

3.1.2 In-session

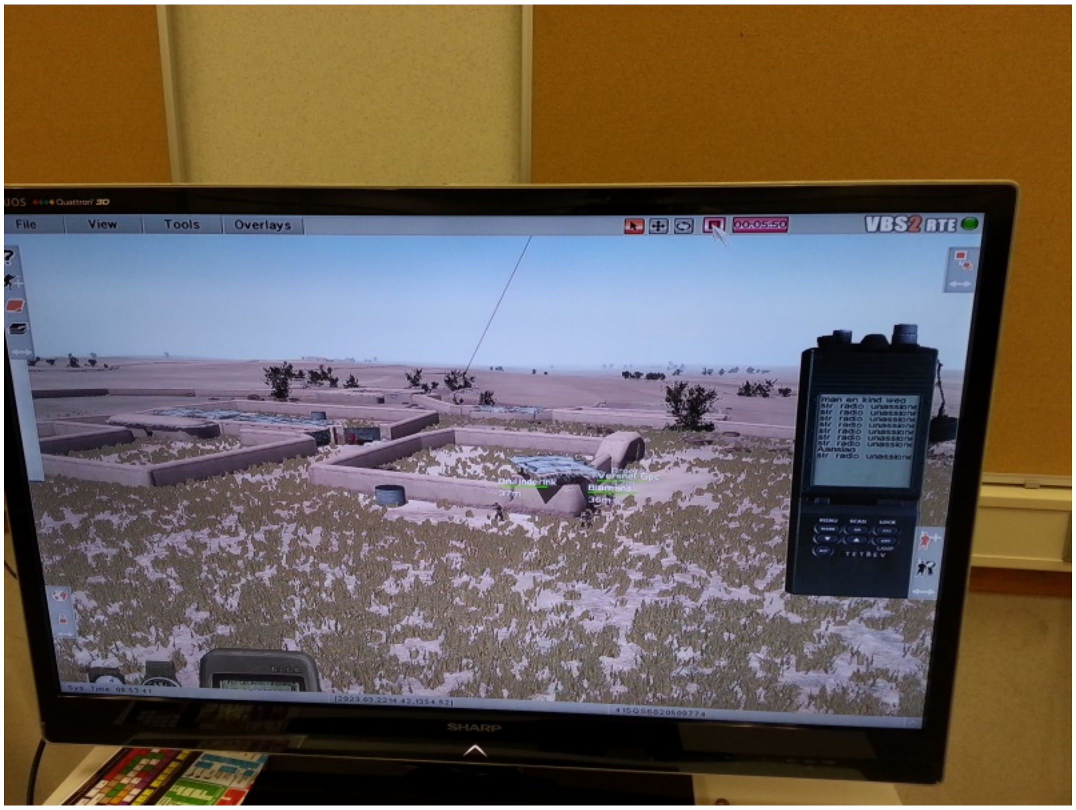


Figure 3.2: A picture of the current situation.

During the training, the instructor needs to use the possibilities provided by the scenario and the serious game to ensure that the trainee receives the optimal learning experience (see figure 3.2 for an overview of the current in-session situation).

In order to achieve this, the instructor has to ensure that all the events have the desired effect. For that, the instructor has to ensure that the events are triggered on time, such that the effect does not miss the trainee. In the broader picture, the instructor has to make sure that the trainee reaches the scenario's learning goals, or makes as much progress as possible. This can be further enhanced by adding events to the scenario during the training. Moreover, the instructor should try to ensure that the trainee stays within the Flow, and is not detached from the game experience during any point in time. Finally, the instructor can make recordings of the scenario and the trainee's behaviour. These recordings can then be used in the After Action Review (AAR).

Currently, the instructor gets an in-game list of all the event indices while executing the scenario (see figure 3.3). As such, the instructor will have to reference the excel sheet if he wants to see any detail of the event. The instructor can trigger each event via this in-game list by simply clicking on the trigger button next to the event index. Besides triggering the existing events, the instructor can also create events on his own within the scenario to ensure that the learning goals are reached or reinforced. This is done by searching for an object in a large list of objects, followed by dragging the object from the list into the game (see figure 3.4). E.g. an enemy soldier. The instructor would then have to assign a way-point to the soldier if the instructor wants the soldier to move. This has to be done for every soldier that the instructor wants present in the scenario.

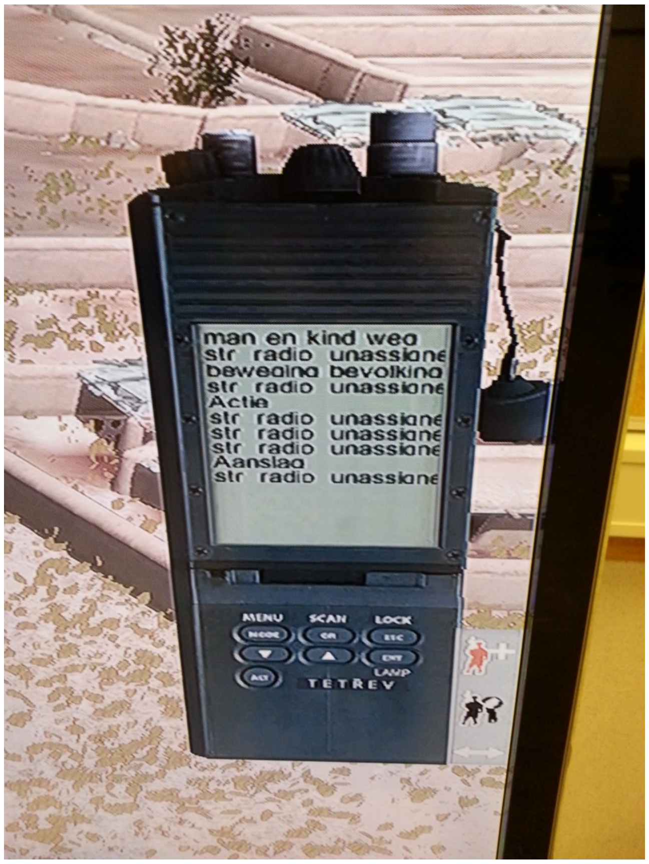

man en kind weg
str radio unassigne
beweging bevolking
str radio unassigne
Actie
str radio unassigne
str radio unassigne
str radio unassigne
Aanslag
str radio unassigne

Figure 3.3: A picture of the in-game event overview in the current situation.

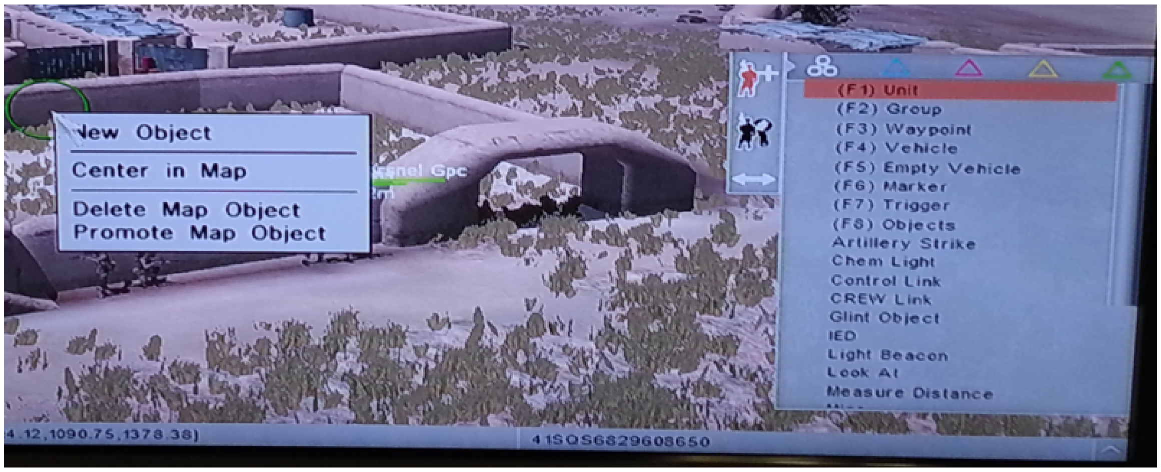


Figure 3.4: A picture of adding an object in the current situation.

3.1.3 Post-session

After the training, the instructor uses the After Action Review to recall the actions of the trainee during the training, and give subsequent feedback on the trainee's performance to ensure that the trainee has learned as much as possible from the training.

3.2 Problems and optimizations

There a number of problems and optimization issues that can be derived from the current situation. In this section these problems and issues will be described.

3.2.1 Pre-session

The scenario being described on an excel sheet, or any kind of document separate from the game, is suboptimal. There is no reason to have disparate environments for studying and executing the scenario respectively in most situations. The instructor should not have to look up an event because only an index is available in the game. Furthermore, having to look up the place where the event occurs on a sheet of paper adds more cognitive load onto the instructor. Due to the fact that it is harder to make a decision when to trigger an event, since the position has to be looked up every time the trainee might be close, whether it is a mental lookup, or looking at the document.

Additionally, events that are related to one another, i.e. reaction events, can be linked more strongly. Currently, the event description will simply mention whether or not the event is related to another event, while a structure could be imposed here.

It would also be more intuitive for the instructor to have hands-on experience with the scenario in-game. This holds especially when it is the instructor's first time, because the instructor would have no frame of reference and could therefore have difficulty linking a scenario on paper to an in-game scenario.

3.2.2 In-session

The issue with referencing events by their index has already been addressed in the previous subsection. Moreover, events already present on the map can't be altered in any way at the moment. The instructor can simply trigger or ignore them, while the option of altering events will give the instructor an easier and more intuitive way of altering the scenario.

Adding events is suboptimal as it is slow and tedious. Having to find the correct object and then dragging the object into the correct position for every object is something that can be optimized. Furthermore, having to control them manually adds a layer of unnecessary micromanagement to the instructor.

3.2.3 Post-session

Due to time and scope constraints, analysis of the post-session tasks was skipped. Moreover, during our observations, it became clear that the pre-session and in-session processes had the most opportunities for improvement.

3.3 Solutions provided by the Instructor Supporting Model

ISM provides solutions for the problems described in the previous section by introducing a graphical user interface, and an A.I component.

3.3.1 Pre-session

ISM allows the instructor to browse through all the events in the game environment before the game starts. The events are placed in the actual game engine and are visually represented by coloured dots. Colours can change per dot, allowing additional information about the event to be displayed through a colour (i.e. different colours would stand for different types of events, orange for triggered events and grey for static events). When selecting an event with, for example, a mouse click, an additional user interface is shown by ISM. This user interface contains the information normally present in the excel sheet. This also means that images of the events' locations are superfluous, as the position of the event is already derived from the in-game view. This results in the excel sheet being redundant in terms of providing event information.

ISM also introduces a structure on events and their reaction events. By allowing the events to be fully described in the game, ISM can embed reaction events within the relevant event, allowing the graphical user interface to display the relevant reaction events when selecting an event.

Because the instructor now uses the game to explore the scenario, he also gets immediate hands-on experience with the game (and the ISM interface). This is particularly helpful for first-time instructors.

3.3.2 In-session

Events can be adapted on the spot before triggering them through ISM, allowing the instructor to make small or big changes to an event before triggering. The instructor can also choose to delete an event.

In order to further assist the instructor in the scenario overview, ISM introduces the possibility of adding a heuristic to an event to determine when it should be triggered. The event then lights up inside the game in order to show the instructor that triggering the event soon is what most likely needs to be done. This also translates to reaction events, allowing ISM to give advice on what event to trigger based on how it thinks the trainee is acting.

By introducing a more intuitive user interface through ISM, the instructor can define how many objects to add when adding an object through the creation of a new event. Moreover, ISM introduces a method of adding A.I. behaviour to these new objects. This allows for more abstract objects to be defined in the event creation, making it easier for the instructor to add more complex events. Because of the A.I. behaviour, controlling the objects to achieve the imagined goal becomes easier, as common goals could be added in the A.I. behaviour list, e.g. move towards the trainee. The A.I. then takes over movement of units automatically and would remove the necessity for instructors to add way points for every type of movement they need.

Since control is crucial, controlling objects remains an optional component. When the instructor chooses to control an object, the A.I. will be immediately disabled until control is released by the instructor.

4 | Implementation

In order to test out the Instructor Supporting Model, we developed a simple game. In this chapter we will discuss the idea behind the game, the environment in which the game was developed, and the details behind the implementation of ISM.

4.1 Game

The Instructor Supporting Model is designed for a serious game in which a training scenario takes place with one or more trainees playing, and one or more instructors supporting the trainee, similar to serious games practised in the Dutch Army. Because the model is also primarily designed for the Dutch Army, the game should be relevant to the game they are currently using: Virtual Battlespace 2 (VBS2).

However, testing the game with ISM requires test subjects. Test subjects within the army are difficult to find, especially in high numbers for a specific week. As a result, the game design had to be changed to accommodate non-military personnel, specifically students, since they are easier to get as test subjects. Moreover, they also share the same age category as most instructors in the Dutch army (early to late twenties). The idea was to translate the idea behind common military serious game scenarios to a game understandable and playable by students.

In most military scenarios, the trainee(s) have to move from their starting position to another position to accomplish a certain task, encountering various events on the way. As such, we made the decision to go for a survival trek game. This type of game requires the trainee to move across a map during which events can take place at certain locations. This way the experience for the instructor should be similar to an instructor supporting a trainee in a military scenario within VBS2.

In the end, two versions of the game are made: a supported one with ISM and an unsupported one without ISM.

4.2 Game engine

In order to decide what kind of game engine would be used to develop the game with and without ISM, we did some cursory research comparing the most prominent game engines. Research was done by browsing gaming communities, testimonials, wikis, knowledge and experience with the use of these engines within TNO, and the game engine sites.

The research was aimed towards several aspects that were either required for the game or would improve the process of development.

First of all, the game engine had to either have the possibility for Local Area Network (LAN) multiplayer, or the possibility for multiplayer to be achieved via a second screen. I.e. a split screen set up where both players would have their own screen and own equipment (mouse and keyboard).

The first method of multiplayer is preferred, since it is easier to develop and allows for a more robust set up.

There was a time constraint on the development time available for this project. As a result, the complexity of the game engine was a predominant factor when it comes to deciding what game engine to choose, since a less complex engine would make it easier, and thereby less time consuming, to develop the game. We define the complexity of an engine by a number of factors:

1. The complexity of the engine in general, whether the GUI is easy to use and how much programming is necessary to achieve a relatively simple game.
2. The programming language, whether there is someone proficient within the "Vergroten Toegankelijkheid van Serious Games" project group (or within TNO) with that language.
3. The community (userbase), whether it is active, extensive, and mature.
4. The official support provided by the developer or through official channels.
5. What kind of extension modules are provided by the community or via the official support channels, and the robustness of these modules.
6. What kind of game bases (basic foundation for a certain type of game) are provided by the community or via the official support channels, and the robustness and relevance of these game bases.
7. The Integrated Development Environments (IDE) that are available for the game engine, and whether we are familiar with one of these environments.
8. The graphical possibilities of the engine, whether it can support 2 dimensional (2D) and/or 3 dimensional (3D). 2D games are easier to program and are also easier to make graphics for in the form of sprites.

We also looked at the performance of the game engine through testimonials. Specifically, we looked at multiplayer performance and possible synchronization issues, since performance issues can break game immersion.

Finally, since only a limited budget is available for developing ISM, cost is taken into account.

Taking all of these factors into account, we looked at the following game engines:

1. GameMaker
2. Source Engine (Source SDK)
3. Unity
4. Unreal Engine 3 - Unreal Development Kit (UDK)
5. VBS2

VBS2 is a primary choice when it comes to relevance to the Dutch military, however, experience within TNO has shown that VBS2 is complex in nature.

As a result of the research done, we ended up picking GameMaker. There is a vast amount of knowledge regarding GameMaker both within the "Vergroten Toegankelijkheid van Serious Games" project and within TNO. Furthermore, the overall complexity of GameMaker was the lowest compared to the other game engines. GameMaker also provided the necessary multiplayer possiblities (LAN multiplayer, specifically) required for instructor-trainee interaction, without performance issues. GameMaker also had a number of existing game bases available that were relevant to our project. On top of all of this, a license for GameMaker was already available within TNO. For the full comparison between the game engines, please refer to appendix A.

4.3 Game design

Instructors in the Dutch army work with a top down overview of the in-game map to refer to event positions. This top-down overview of the map contains all the relevant information of 3D representation has, since the core concept is the movement between two points on this map, during which certain events would occur. Therefore, it is possible to execute the same type of game with a 2D top-down visualization, albeit with less realism. But realism is not important for the instructor, as he only needs to have a good overview. As such, we can restrict our game to a top-down game.

A survival trekking game done in a 2D top-down style can be implemented in a variety of ways through the following game concepts: a turn based strategy (TBS) type of approach, a real-time strategy (RTS) type of approach, and a top-down shooter type of approach.

In a turn based strategy game, players take turns, usually moving multiple units around to defeat one another. TBS games are primarily controlled using a mouse. As a result, a TBS style of game allows the instructor to control a multitude of units by design. However, TBS does not capture the nature of time-dependent decision making, as it is turn based. As such, it is much harder to capture the support provided by ISM to allow for easier and faster decision making.

In a real-time strategy game, players move at the same time without turns, usually moving multiple units around to defeat one another. RTS games are primarily controlled using a mouse. As a result, an RTS style of game allows the instructor to control a multitude of units by design. RTS also allows for the possibility of time-essential decision making.

In a top-down shooter, players move at the same time without turns, controlling a single unit to defeat one another. Top-down shooters are primarily controlled using a keyboard. As a result, controlling a multitude of units is not part of the design of a top-down shooter. This complicates events that feature more than one unit and complicates unit control for the instructor in general.

Therefore, we decided to take an RTS game base. Specifically, we took the “Awesome 2d Rts Engine” by “rude guss” ((rude guss, 2009)). This base also has a basic multiplayer component as added bonus, which can be extended at will.

4.4 Game concept

The game is a savannah survival game. There are two persons involved in playing the game: 1 trainee and 1 instructor. The trainee controls a single trainee unit that he has to safely navigate to the finish while the instructor has to ensure that the trainee learns as much as possible.

On the way, the trainee can encounter a number of events:

1. **Trigger events**
   These events are triggered and controlled by the instructor and should be triggered based on the actions of the trainee.
2. **Static events**
   These events are already present in the scenario, and will therefore not need to be triggered by the instructor, as they are already active.

3. **Reaction events**
   These events are linked to either a trigger or a static event. They are triggered and controlled by the instructor and should be triggered based on a trainee's reaction to the linked event. If the trainee's reaction is close to the expected behaviour, then the reaction event should be triggered.

An event consists of one or more instances of a single object. There are a number of objects present in the game:

1. **Trainee**
   The trainee is the object (unit) controlled by the trainee player. No other instances of the trainee can or will appear. The trainee has a health bar that will go down when a lion attacks it. When the health bar reaches 0, the trainee will be dead. The trainee can take a total of 20 lion bites.

2. **Finish**
   The finish is the object the trainee needs to touch to successfully complete the scenario. The finish is static and cannot be controlled.

3. **Time**
   The timer starts counting down from 300 when the scenario starts. If it reaches 0, the trainee has failed to reach the finish in time and the scenario will end.

4. **Lions**
   Lions can attack the trainee and thereby drain the trainee's health. They will attack the trainee if he comes too close and only move back when he is standing still.

5. **Lion pits**
   Lion pits are home to the lions. If the trainee comes too close to a lion pit, a number of lions will come out.

The instructor has to ensure that the events are triggered at the correct time and control the lion and lion pits. He can also create events on his own. The instructor's eventual goal is to train the trainee as much as possible in these learning goals:

1. The trainee has to recognize and avoid lion pits (a lion pit contains one or several lions that appear and challenge the trainee when he comes too close to the pit).

2. The trainee has to recognize lions, and stay at a safe distance from them.

3. The trainee has to learn how to handle lions when they are chasing him (if the trainee stands still, the chasing lions will stop and return to their habitat).

4.5 Game implementation

The RTS game base consists of a number of features that represent the RTS genre. In multiplayer, every player has control of a number of units and buildings. Every unit can walk, gather resources, attack enemy units or buildings, and even cast spells. You control units by selecting them and then moving them with the mouse. Units can also be destroyed. Buildings can create units using the resources available. Buildings can also be destroyed. It is possible to select multiple units or buildings by dragging the mouse to make a selection box spanning the units or buildings you wish to control.

Every unit or building has a number of variables. These variables can change per unit/building but are predefined per type of unit/building. For example, a unit has a certain number of starting hit points (the amount of life a unit has, if it reaches 0, the unit is destroyed), a certain move speed, a certain attack speed, a certain attack range, and a certain damage done per attack. A lot more variables are defined per unit, but we will not go over all of them, as they are not very relevant to ISM. Furthermore, a unit also has a certain sprite (a 2D image) that represents how that unit looks in the game.

An example of variables that are predefined per unit type:

```
unit_hp_max = 20; // the maximum amount of hit points the unit can have
unit_hp = 20; // the amount of hit points the unit has

unit_speed = 2;

attack_cooldown = 45;

autoattack_range = 0;

unit_damage = 1;
```

Furthermore, every few milliseconds, every unit and building get a step event. In this step event, the unit/building can do an action based on some of the current variables set within the unit/building. For example, IF a unit has a) A certain position it should move towards. b) Is not at that position yet. c) Can move. And d) No other command has been given. THEN the unit should move towards that position. Alternatively, IF a unit has a) An attack target. b) The unit can attack. c) The enemy unit is still alive and in range of the attack. And d) No other command has been given. THEN the unit should attack that target.

An example of variables that decide what kind of action a unit will execute:

```
able_to_move = true;
move_x = 10;
move_y = 8;

attack_mode = false;
attack_target = 99; // the id of the target is set to 99, a non-existing
target
```

All the units and buildings are positioned on a map. This map is made using the GameMaker room editor. Units and buildings are placed on the map using the editor. Other objects, such as trees, are also placed using the room editor. The trees are the type of objects that make up the physical boundaries of the map. E.g. a row of trees would act as a wall. Before the game starts, there is a multiplayer lobby in which the players can chat and where one of the created maps is selected as the map on which the game will be played.

Multiplayer is achieved using the “Faucet Networking” extension package for GameMaker. In this extension, network communication is done by writing bytes of data from client to client over a TCP connection. In multiplayer, there is a server and one or more clients connected to the server, every one running their own instance of the game. All the clients send their data to the server, as the server acts as a hub. The server then transmits the necessary data to all the clients, so the clients are always up-to-date. The server also runs the game, and is therefore also a client, but since it also acts as a server, it can bypass sending the appropriate data to itself and instead handle this data internally.

During the game, the player will only see the parts of the map on which his units are present. The rest of the map is covered by something called *fog of war*. This fog of war is a visual representation of the uncertainty of what is happening on the part of the map that is not controlled by the player. This is visualized by blacking out the parts of the map on which the player has no units or buildings.

A heads-up display (HUD) is also present to give additional information and control to the player. The HUD shows the special abilities (if any) of the units and buildings. It also has a minimap that shows the entirety of the map (with fog of war blocking off the appropriate parts) in a small format. The HUD is present on the bottom of the screen.

4.6 ISM implementation

4.6.1 Trainee

The trainee is a special type of unit since it is the centrepiece within the game. The trainee has the following special properties:

1. The trainee is not able to attack.
2. The trainee is slower than lions.
3. The trainee is able to take a specific number of lion hits, specifically 20.
4. The trainee is tracked inside the game to ensure that the A.I. always knows where the trainee is. It is also used to determine when the trainee has died.
5. The trainee's hit points are always visible.
6. When the trainee dies, the scenario is over.

In order to ensure that the trainee does not attack the lions, the attack range has been set to -1 and the attack damage has been set to 0:

```
autoattack_range = -1;
unit_damage = 0;
```

To ensure that the trainee is slower than lions, the movement speed of the trainee has been set to 1.5:

```
unit_speed = 1.5;
```

By setting the trainee HP to 20 (and the lion's attack damage to 1), we ensure that the trainee can take 20 lion hits:

```
unit_hp_max = 20;
unit_hp = 20;
```

To keep track of the trainee within the appropriate in-game objects, the trainee's id is stored within a global variable. This way, any object can always query this global variable to get any information regarding the trainee, or to alter the trainee in any way:

```
global.trainee = id;
```

4.6.2 Lions

Lions are the units that the trainee interacts with. Lions have a number of properties to ensure that they interact correctly with the trainee. Furthermore, lions also have an A.I. component that is only active in the supported version of the game. Lions have the following special properties:

1. Lions are faster than the trainee.
2. The trainee is able to take a specific number of lion hits, specifically 20.

To ensure that lions are faster than the trainee, the movement speed of lions has been set to 2 (in contrast to the trainee's movement speed of 1.5):

```
unit_speed = 2;
```

Since the trainee has 20 HP and should be able to take 20 lion hits, the damage for lions has been set to 1 damage per hit:

```
unit_damage = 1;
```

The lion's A.I. module ensures that the lion chases the trainee when he comes too close, and will continue to chase for as long as the trainee is moving. As soon as the trainee stops moving, the lion should move back to its place of origin.

First of all, when the lion comes into existence, it registers its current location as its point of origin:

```
origin_x = x;
origin_y = y;
```

Then, at every game step, the A.I. simply checks what needs to be done if the instructor is supported. There are two behaviour modes: the lion is either attacking because the trainee moved too close, or is relaxed because the trainee is not moving in range. Alternatively, the instructor can assign either of these behaviours to a lion when adding an event. When the lion is attacking, it queries the position of the trainee through the global trainee variable and will decide whether the trainee has moved from his previously known position. If the trainee is still at his previously known position, the lion will start moving back to his place of origin. Otherwise, it will move closer to the trainee (i.e. start chasing) and attack it if its in range. When the lion is relaxed, it will do nothing until a trainee comes close, in which case it will start chasing. Finally, as soon as an outside command is issued (i.e. by the instructor), the lion will ignore all commands given by the A.I. module:

```
if global.supported
{
    switch(behaviour)
    {
        case "attack": // if the lion is in attack mode

            // if the trainee is still at the same position (hasn't moved)
            if global.trainee.x == prev_trainee_x
            && global.trainee.y == prev_trainee_y
            {
                attack_mode = false; // deactivate attack mode
                attack_target = 99; // no attack target
                able_to_move = true;
```

```
                 // move back to original position
                 move_x = origin_x;
                 move_y = origin_y;

                 behaviour = "relaxed"; // go to relaxed mode
             }
             else
             {
                 attack_target = global.trainee;
                 attack_mode = true;
                 able_to_move = false;

                 // move to trainee
                 move_x = global.trainee.x;
                 move_y = global.trainee.y;
             }
         break;

         case "relaxed": // if the lion is in relaxed mode

             if distance_to_object(global.trainee) < 60 // trainee comes too
close
             {
                 if move_x != x || move_y != y // if the lion is not moving
                 {
                     // if the trainee has moved
                     if global.trainee.x != prev_trainee_x
                     || global.trainee.y != prev_trainee_y
                     {
                         behaviour = "attack"; // go to attack mode
                     }
                 }
             }
         break;
     }

     // update the trainee's previous known position
     prev_trainee_x = global.trainee.x;
     prev_trainee_y = global.trainee.y;
 }
```

Lions can also have initial behaviour:

1. **Stay**
   The lion stays at his starting position, this is the default.
2. **Approach**
   The lion instantly starts moving to the position of the trainee.

4.6.3 Lion pits

Lion pits are the buildings that the trainee interacts with. Lion pits have a number of properties to ensure that they interact correctly with the trainee. Furthermore, lion pits also have an A.I. component that is only active in the supported version of the game. Lion pits have the following special properties:

1. Lion pits do not collide with any unit.
2. Lion pits only release lions once.

In order to ensure that lion pits do not collide with other units (such as lions or the trainee), it is made solid within GameMaker. To ensure that lions only spawn once, a boolean is created to keep track of this:

```
lions_spawned = false;
```

The lion pit's A.I. module ensures that a number of lions come out of the lion pit when the trainee comes too close. At every game step, the lion pit simply check whether the trainee is within range of the lion pit by querying the trainee's position and comparing that with its own. Then, if the trainee comes too close, it will create the appropriate amount of lions a single time. The amount of lions it should spawn is predefined by the behaviour:

```
if global.is_instructor && global.supported
{
    // if the trainee is close, and the lions haven't come out yet
    if distance_to_object(global.trainee) < 70 && lions_spawned = false
    {
        lions_n = 0;

        // determine the amount of lions to spawn from the object's behaviour
        switch(behaviour)
        {
            case "spawn 1 lion":
                lions_n = 1;
            break;

            case "spawn 2 lions":
                lions_n = 2;
            break;

            case "spawn 3 lions":
                lions_n = 3;
            break;
        }
        eventExec(x, y, obj_lion, lions_n, "approach", "auto");
        lions_spawned = true;
    }
}
```

4.6.4 Finish

The finish is a building that the trainee interacts with by getting close to it. The finish has a number of properties to ensure that it interacts correctly with the trainee. The finish has the following special properties:

1. The finish cannot be moved or controlled.
2. When the trainee is in range of the finish, the scenario ends.

In order to ensure that the finish cannot be controlled or moved by any player, it is made as a neutral building. Since buildings cannot be moved and neutral units cannot be attacked.

The finish checks at every game step if the trainee is close enough to the finish for the scenario to end:

```
// if the trainee touches the finish
if distance_to_object(global.trainee) < 1 && !ended
{
    ended = true;
    endScenario("Trainee has finished! Scenario is over!");
}
```

4.6.5 Timer

The timer is a counter that counts down from 300 to 0 (in seconds). It is always displayed at the top left of the screen. It has a singular property:

1. When the timer reaches 0, the scenario ends.

The timer checks at every game step if it has reached 0, if it has, it will end the scenario:

```
if time_left == 0
{
    if global.timed_end
    {
        endScenario("Time is up! Scenario is over!");
    }
}
else
{
    alarm[0] = 30;
    time_left -= 1;
}
```

4.6.6 Events

Events are special objects that are derived from building objects. Every unit can pass through an event, and events are only visible to the instructor if the instructor is playing the version supported by ISM. If the instructor is unsupported, the events are simply listed in text form in the HUD. How the event appears for the two versions of the game will be handled in the next section *Heads-up display.*

First of all, an event has a single textual description:

```
description = "If the trainee has not tried the other path, 1 lion spawns.";
```

Then the type of event is defined, i.e. whether it is static or triggered:

```
is_static = false;
```

An event consists of a type of object, a number denoting the amount of objects, and a location in which the event takes place. If the event is static, these are not used, as the event is already integrated in the game map:

```
object_to_create = obj_lion;
objects_n = 1;
spawn_x = 22;
spawn_y = 16;
```

Furthermore, these objects can also have a type of behaviour, which is only used if the version is supported by ISM. The way the behaviours are handled differs per object. Lions can have stay and approach behaviour, stay means that the lion will stay at his position until a trainee comes close, approach means he will go directly to the trainee, no matter how far away he is. Lion pits have the amount of lions that they spawn as behaviour:

```
object_behaviour = "stay"; // in this case, we tell the object to have the
"stay" behaviour
```

The event can contain a number of expected behaviour types that might be shown by the trainee in response to this kind of event. First, the number of expected behaviour types is defined:

```
expectedbehaviour_n = 1;
```

Following this, every expected behaviour type is defined by a textual description and whether this type of behaviour is deemed to be correct:

```
expectedbehaviour_text[0] = "Trainee tries to force its way through.";
expectedbehaviour_correct[0] = false;
```

For the A.I. module within the ISM supported version, the appropriate heuristic for determining whether the trainee is showing this kind of behaviour type is also defined (for more information regarding this implementation, please refer to the *Trainee prediction* sub-section):

```
expectedbehaviour_trigger[0] = "trainee_close";
```

For every expected behaviour type, the appropriate response event can also be defined, which should be triggered by the instructor if the trainee shows the expected behaviour. The response event is defined similarly to how a normal event is defined; with a type of object, a number denoting the amount of objects, a location in which the event takes place, and an optional type of behaviour for the supported version:

```
expectedbehaviour_response_object_to_create[0] = obj_lion;
expectedbehaviour_response_objects_n[0] = 2;
expectedbehaviour_response_spawn_x[0] = 30;
expectedbehaviour_response_spawn_y[0] = 20;
expectedbehaviour_response_object_behaviour[0] = "stay";
```

When an event (normal or response) is triggered, for every object, the following is done:

```
for(i = 0; i < objects_n; i += 1)
{
```

The exact position of the individual object is calculated from the event's internal position. This is done to ensure that objects do not spawn on top of each other and collide:

```
// we divide the objects and place them separately, so we don't get collisions

// if it's part of the first half, we place the object on the left
if (i <= (objects_n / 2))
{
    exact_x = spawn_x - (25*i);
    exact_y = spawn_y ;
}
// if it's part of the second half, we place the object on the right
else
{
    exact_x = spawn_x + (25 * (i - (objects_n / 2)));
    exact_y = spawn_y ;
}
```

Then the object type and object behaviour (if defined) is translated into a numeric identifier so that it can be transmitted over the network (e.g. the lion object gets the identifier '1'):

```
switch(obj_to_create)
{
    case obj_lion:
        obj_id = 1;
    break;

    case obj_lionpit:
        obj_id = 2;
    break;
}

switch(behaviour)
{
    case "stay":
        behaviour_id = 1;
    break;

    case "approach":
        behaviour_id = 2;
    break;
}
```

Note: not all identifier translations were included here.

Finally, the object data is transmitted to the server:

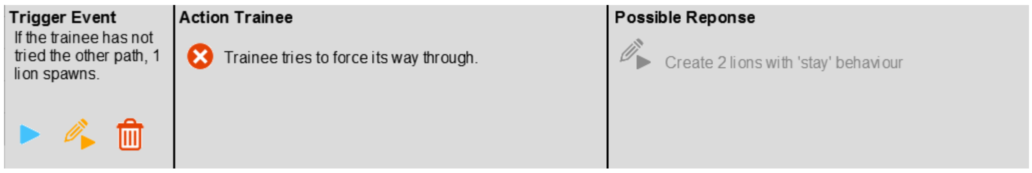
```
write_ushort(global.ServerSocket, 2000);
write_ushort(global.ServerSocket, obj_id);
write_ubyte(global.ServerSocket, global.PlayerId);
write_short(global.ServerSocket, exact_x);
write_short(global.ServerSocket, exact_y) ;
write_ushort(global.ServerSocket, behaviour_id);
```

This is repeated until all the objects of which the event consists are send to the server.

4.6.7 Heads-up display

The heads-up display (HUD) is always present on the bottom of the screen. The HUD shows different things depending on whether the supported or the unsupported version is being used. As such, this section is divided into two parts, each part showcasing one of the cases.

Game supported by ISM



Trigger Event
If the trainee has not tried the other path, 1 lion spawns.

Action Trainee
Trainee tries to force its way through.

Possible Reponse
Create 2 lions with 'stay' behaviour

Figure 4.1: A picture of the HUD in the game supported by ISM.

The HUD is divided into three parts: left, centre, and right. The HUD will remain empty until an event is selected by the instructor:

```
 if global.number_of_selected = 1 && instance_exists(obj_eventHUD.target)
 {
     if obj_eventHUD.target.is_event
     {
```

When an event is selected, the HUD will show the textual description of the event on the left side of the HUD, together with the buttons to trigger, adapt, or delete the event. Adapting an event will show a custom event pop up (see the *Custom event* sub-section) with the variables filled in to fit the selected event, so the instructor can alter the event and then trigger it.

The centre of HUD showcases the possible trainee reaction types. The possible reactions are described via their respective textual descriptions, together with an icon showcasing whether the behaviour is correct or faulty:

```
 if obj_eventHUD.target.expectedbehaviour_correct[i]
 {
     // draw the correct action sprite at a certain position in the HUD
     draw_sprite(spr_correct_action,
     /*-1, view_xview[0]+actionwindowx,
view_yview[0]+actionwindowy+2+(50*i)*/);
 }
 else
 {
     // draw the faulty action sprite at a certain position in the HUD
     draw_sprite(spr_faulty_action,
     /*-1, view_xview[0]+actionwindowx,
view_yview[0]+actionwindowy+2+(50*i)*/);
 }
```

The reaction events are on the right side of the HUD, and vertically correspond to the appropriate trainee reaction (i.e. every possible trainee action has the reaction event directly to the right). Every reaction event is textually described by parsing the amount of objects, the object that is to be created, and its behaviour:

```
 name = "Create " + string(objects_n) + " " + obj_to_create_name + "s" + "
with '" + behaviour + "' behaviour";
```

Every reaction event also has a trigger and an adapt button, which work in the same way as they do for a normal event. However, they are disabled if the original event has not yet been triggered. Furthermore, the text will also be greyed out. This is to prevent the instructor from accidentally mistaking one of the reaction event's trigger buttons for the main event trigger button, something that we observed as happening quite often during test runs:

```
 if is_active
 {
      // draw the adapt event button at a certain position in the HUD
      draw_sprite(spr_adapt_event,
      /*-1, view_xview[0]+xpos, view_yview[0]+ypos*/);
 }
 else
 {
      draw_set_alpha(1);
      draw_set_color(c_gray);
 }
```

Finally, a rectangle is drawn around the most likely reaction to have occurred based on the trainee prediction mechanism of ISM (see subsection 4.6.10):

```
 // if the prediction score is above the threshold of 40
 if obj_eventHUD.target.most_probable_score > 40
 && i == obj_eventHUD.target.most_probable_index
 {
      // draw a rectangle around the predicted action
      draw_rectangle(/*view_xview[0]+actionwindowx,
view_yview[0]+actionwindowy+(50*i),
      view_xview[0]+actionwindowx+380, view_yview[0]+actionwindowy+(50*i)+50,
true*/);
 }
```

Game not supported

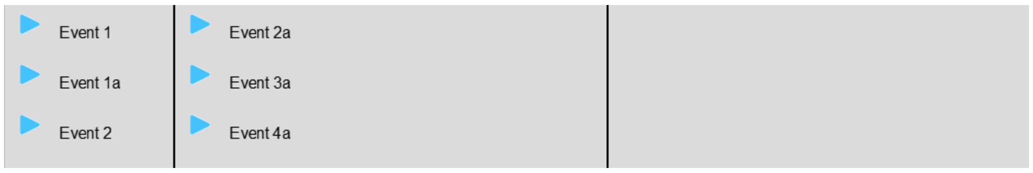


Figure 4.2: A picture of the HUD in the game not supported by ISM.

When the game is unsupported, the HUD is used to display all the events within the game. Every event is textually described via the word “Event”, combined with a number and a letter if it is a reaction event. As such, the first event in the map would be called “Event 1”, and the second event would be called “Event 2”:

```
 trigger.alt_info = "Event " + string(i+1);
```

Two reaction events to the first event would then be called “Event 1a” and “Event 1b” respectively. We use chr(97) (’a’) here as a base, so we can automatically calculate the next letter from that position using the hexadecimal translation of the character (e.g. chr(97+2) would be ’c’):

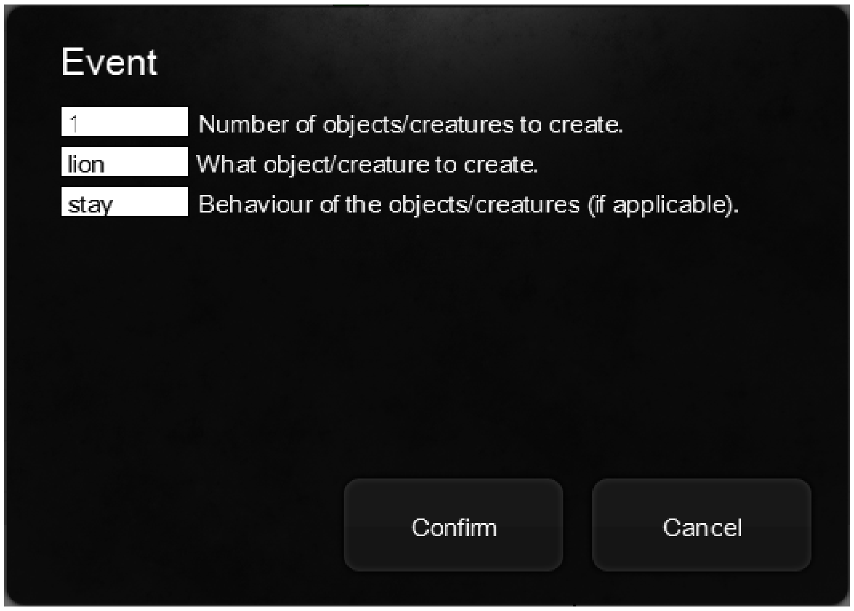
```
response.alt_info = "Event " + string(i+1) + chr(97+j);
```

Every of these events would have a trigger button next to them, which is used by the instructor to trigger the events.

4.6.8 Custom event

By pressing the middle mouse button, the instructor is able to add his own event to the map at any point in time. When the middle mouse button is pressed, a pop-up screen appears in which the instructor can specify an event by supplying the following information through drop-down boxes (see figure 4.3):

1. Number of objects to create (only in the version supported by ISM).
2. Type of object to create.
3. Behaviour of these objects (only in the version supported by ISM).



Event

1 Number of objects/creatures to create.
lion What object/creature to create.
stay Behaviour of the objects/creatures (if applicable).

Figure 4.3: A picture of the pop-up screen in which the instructor can specify a custom event.

The number of objects to create can differ between 1 and 5 and there are two types of objects that can be created: lions and lion pits.

Finally, the drop-down box for the behaviour changes depending on what type of object is selected. If lion is selected for the object type, stay and approach are available as behaviours. Conversely, if lion pit is selected for the object type, spawn 1 lion, spawn 2 lions, and spawn 3 lions are available as behaviours. We use the dm_create_menu function (an open source drop-down box function) to create a drop-down box, and then add options to the drop-down box using ddm_add_option:

```
switch(obj_to_create_name)
{
    case "lion":
        ID_BEHAVIOUR = ddm_create_menu(x-230, y-75, "stay", 10, false, false,
msgBox_Font);
        ddm_add_option(ID_BEHAVIOUR, "stay");
        ddm_add_option(ID_BEHAVIOUR, "approach");
    break;

    case "lion pit":
        ID_BEHAVIOUR = ddm_create_menu(x-230, y-75, "spawn 1 lion", 10,
false, false, msgBox_Font);
        ddm_add_option(ID_BEHAVIOUR, "spawn 1 lion");
        ddm_add_option(ID_BEHAVIOUR, "spawn 2 lions");
        ddm_add_option(ID_BEHAVIOUR, "spawn 3 lions");
    break;
}
```

4.6.9 Trainee vision

In order to assist the instructor in appropriately adding or altering events on his or her on behest, the game with ISM offers the instructor the possibility of seeing what the trainee sees through an enhancement of the fog of war has been introduced. This enhancement shows the instructor a transparent version of the fog of war the trainee experiences.

First, ISM checks whether the instructor is playing the game and if the current version is supported by ISM, if these two factors are true, the fog variable is created:

```
// we want to create fog for the trainee and for the supported instructor.
if global.is_trainee || (global.is_instructor && global.supported)
{
    x = view_xview[0]+(16+(i*32));
    y = view_yview[0]+(16+(z*32));
    dist[i,z] = distance_to_object(global.trainee);
    if dist[i,z] <= view
    {
        fog[i,z] = false;
    }
    else
    {
        fog[i,z] = true;
    }
}
```

Finally, when drawing the fog, if the instructor is unsupported by ISM, simply no fog layer is drawn. Otherwise, if the instructor is supported by ISM, a specific alpha is applied to the fog layer to ensure a degree of transparency that does not pose a hindrance for the instructor. As such, the instructor can still see the fog of war, but still see everything. The code behind this is rather technical, and as a result, we have moved it to the appendix. Please refer to appendix B for this piece of code.

4.6.10 Trainee prediction

Trainee prediction is used to determine the most relevant event at a point in time. The heuristic used to determine the most relevant normal event is as follows:

1. Enumerate through all events that have not been triggered.
2. Take the event closest to the trainee.
3. See if the trainee has not already passed this event.
4. If not, check if the trainee is within a static radius of the event.
5. If the trainee is within this radius, highlight the event.

For reaction events, it becomes impossible to make a general heuristic, as the expected behaviour of the trainee would have to be checked through the heuristic, which is much too complex for a single heuristic, as the trainee's behaviour could be anything. It is therefore impossible to define a singular heuristic. As a result, the heuristics would have to be defined per expected behaviour, as certain specific types of behaviour can be defined for each event. These heuristics should be defined for the most common behaviours:

1. Get the active event.
2. For every reaction event, run the corresponding heuristic on the trainee.
3. The reaction event with the highest result gets highlighted.

In the game, the heuristics for two types of behaviours are defined:

1. Whether the trainee is close to the position (location) of the event. I.e. the trainee is getting too close to either a group of lions, or a lion pit.
2. Whether the trainee has passed the position (location) of the event. I.e. the trainee has successfully circumvented a group of lions, or a lion pit.

These heuristics can be used to loosely determine whether the trainee has responded correctly to the events in the game. Since every event in this game represents lions or lion pits, getting close to an event is faulty behaviour. As such, we can use the heuristic of how close a trainee is to judge his behaviour. On the other hand, if the trainee has passed the event successfully, he most likely showed correct behaviour by avoiding the lions or lion pits.

These heuristics result in a score to determine which behaviour the trainee is showing. The scores from both heuristics are then compared to see which one is more prevalent in representing the trainee's behaviour. If the highest score exceeds the predetermined threshold (0), the behaviour will be visually represented as the trainee's behaviour in the HUD. In order to determine how close the trainee is to the event, the distance between the trainee and the event is taken. To determine whether the trainee has passed the event, the distance between the trainee and the finish, and the event and the finish is compared. If the trainee is closer to the finish than the event is to the finish, the trainee has successfully passed the event. However, only if the distance between the trainee and the event is big enough will the trainee passed heuristic overrule the trainee close heuristic, since the trainee could still come too close to the event if he makes a wrong move:

```
for(i = 0; i < expectedbehaviour_n; i = i+1)
{
    switch(expectedbehaviour_trigger[i])
    {
        // run the trainee_passed heuristic
        case "trainee_passed":
                // if the trainee is close to the finish than the event is to the
   finished, the trainee has passed
                if distance_to_object(global.finish) >
                point_distance(global.trainee.x, global.trainee.y,
   global.finish.x, global.finish.y)
                {
                    // use the distance from the event to the trainee in the
   score because the trainee might still be close enough for the trainee_close
   heuristic to be relevant
                    behaviour_score = 80 + distance_to_object(global.trainee);
                }
                else
                {
                    // we set the score to an arbitrary low number
                    behaviour_score = -100;
                }
            break;

            // run the trainee_close heuristic
            case "trainee_close":
                // the closer the trainee is to the event, the higher the score
                behaviour_score = 100 - distance_to_object(global.trainee);
            break;
        }

        // set the most probable behaviour from the heuristic scores
        if behaviour_score > most_probable_score
        {
            most_probable_score = behaviour_score;
            most_probable_index = i;
        }
    }
```

4.6.11 Statistics

A number of in-game statistics are recorded for the purpose of data analysis. Specifically, the following data is recorded:

1. Total events created.
2. Total events adapted.
3. Total events triggered.
4. Per event: whether it was triggered.
5. Total number of lions present at the end of the scenario.
6. Total number of lion pits present at the end of the scenario.
7. Total number of left mouseclicks.
8. Total number of middle mouseclicks.
9. Total number of right mouseclicks.
10. Time left.

11. Trainee health left

These statistics are saved into a data file at the end of a scenario. The datafile is named after the player combined with the date/time and whether the version was supported by ISM:

```
 fileid = file_text_open_write(dir + global.ClientName + " - " +
string_replace_all(date_datetime_string(date_current_datetime()), ':', '') +
".txt");
```

5 | Experiment

In order to test whether the Instructor Supporting Model helps the instructor in creating the optimal learning experience, we did an experiment with the prototypical game we developed. In this experiment, the test subject plays the role of the instructor, while the experiment leader plays the role of trainee with the help of a script. In this chapter we will discuss our hypotheses, the experimental set up, and the experiment itself.

5.1 Hypotheses

We want to know whether the game with the Instructor Supporting Model provides better support for the instructor than the unsupported game when it comes to creating the optimal learning experience. In order to test this, we set up a number of hypotheses to test the key improvements and pitfalls:

- **Hypothesis 1**:
  With ISM the instructor will have a better overview of the scenario.

  ISM offers the opportunity to preview the events in the in-game environment. This should allow for the instructor to get a realistic overview of the scenario and as a result a better feel for the scenario before it starts, and while the scenario is being played.

- **Hypothesis 2**:
  With ISM the instructor will plan more often before the scenario starts.

  Since ISM should allow the instructor to have a better overview of the scenario before it starts, the instructor should plan more often, i.e. have a an idea of the actions he could take in response to certain trainee behaviour. As the scenario preview is similar to the actual scenario, the instructor does not have to mentally translate certain objects to their real-time equivalents. As such, it should be easier for the instructor to translate the observations made during the preview to the actual game, allowing for easier planning.

- **Hypothesis 3**:
  With ISM the instructor will make adjustments more easily in the scenario. Therefore the instructor will make more adjustments on average.

  The interactive events introduced by ISM eliminate the need to read the events' descriptions on paper. Furthermore, ISM allows the instructor to specify adjustments more easily due the instructor being able to specify the number of objects present introduced by the event rather than creating a new event for every object. Finally, the A.I. elements introduced by ISM should partially relieve the mental workload of the instructor as the instructor has to focus less on object movement and behaviour. This all should result in it being easier for the instructor to consider and create his own events during the game.

- **Hypothesis 4**:
  ISM will result in the challenge level of the scenario being more appropriate for the trainee.

With the support granted by ISM, the instructor should be able to adjust the scenario better. As a result, the difficulty should decrease for inexperienced trainees and conversely, increase for experienced trainees. This should result in an appropriate challenge level for the trainee, resulting in an increase of the inexperienced trainees' performance, and a decrease of the experienced trainees' performance.

- **Hypothesis 5**:
  ISM will be accepted by the instructor because it is usable and because it is technologically sound.

  ISM should be both intuitive and usable by the instructor. No programming errors should hinder the instructor's experience.

- **Hypothesis 6**:
  ISM results in the instructor having a higher opinion of his own performance in triggering, adapting, and adding events.

  Due to the support ISM provides, the instructor should have a better understanding of how to handle events in the scenario. As such, the instructor will feel more in control and should have a higher opinion of his own performance.

5.2 Measures

In order to test the hypotheses, we introduce a number of measures to gather the required statistics. We used the following measures during the experiment:

- **In-game statistics**
  These statistics are gathered in the prototypical game while the scenario is being ran (see 4.7.11 for more info). The statistics are then output in textual form.

- **Screen capturing**
  The computer screen of the test subject was being recorded with the use of BB FlashBack by Blueberry Software. This recording was then used to classify certain data from the in-game statistics. Specifically, it was used to differentiate between adding necessary events (in the unsupported version) and adding custom events. Furthermore, it was also used to fill in incomplete or corrupt data gathered from the in-game statistics by going over the recording and filling in the data manually.

- **Questionnaire**
  At the end of the experiment, we had the test subject fill out a questionnaire to get data on the test subject's experiences.

For every hypothesis, there is one or more measure in place to provide the data necessary to test the hypothesis:

- **Hypothesis 1**:
  We use the questionnaire to ask whether the test subject had a good overview of the scenario before it started.

- **Hypothesis 2**:
  We use the questionnaire to ask whether the test subject had a plan before the scenario started.

- **Hypothesis 3**:
  We use the in-game statistics to determine how many events were added to the scenario by the test subject. Furthermore, we also check how many events were adapted by the test subject.

- **Hypothesis 4**:
  We use the in-game statistics to see how many hit points and how much time the trainee had left at the end of the scenario. We use this to judge how difficult the scenario was for the trainee.
- **Hypothesis 5**:
  We use the questionnaire to ask whether the game behaved according to the test subject's expectations. We also ask whether the interface was easy to use.
- **Hypothesis 6**:
  We use the questionnaire to ask whether the test subject was able to correctly and timely trigger or add events, according to his own opinion.

5.3 Test subjects

We recruited the test subjects via e-mail and personal contact. We send out an e-mail to all the interns at TNO with the request to participate in the experiment. We personally approached some of our own contacts with the same request.

Our criteria was that the test subject was a bachelors, or masters student. Additionally, we required the subject to be at least 18 years of age and at most 32 years of age. We also looked to have a variety of both female and male test subjects, so that the test was not too dominated by a single gender.

5.4 Experimental set-up

The experiment is done with two groups of test subjects. The supported group does the experiment with the game supported by ISM, the unsupported group does the experiment with the unsupported game. We can then use the data collected via the measures to compare between groups and thereby testing the hypotheses.

Most of the experiments (15/17 test subjects) were done in at a TNO office, the remainder (2/17) was done in a room in a student house.

The training sessions were performed using two computers linked with an Ethernet cross-cable. One computer is used by the test subject (instructor), the other by the experiment leader (trainee). The computers were on opposing sides, so the participants could not peek at each other's screens (see figure 5.1 and figure 5.2).

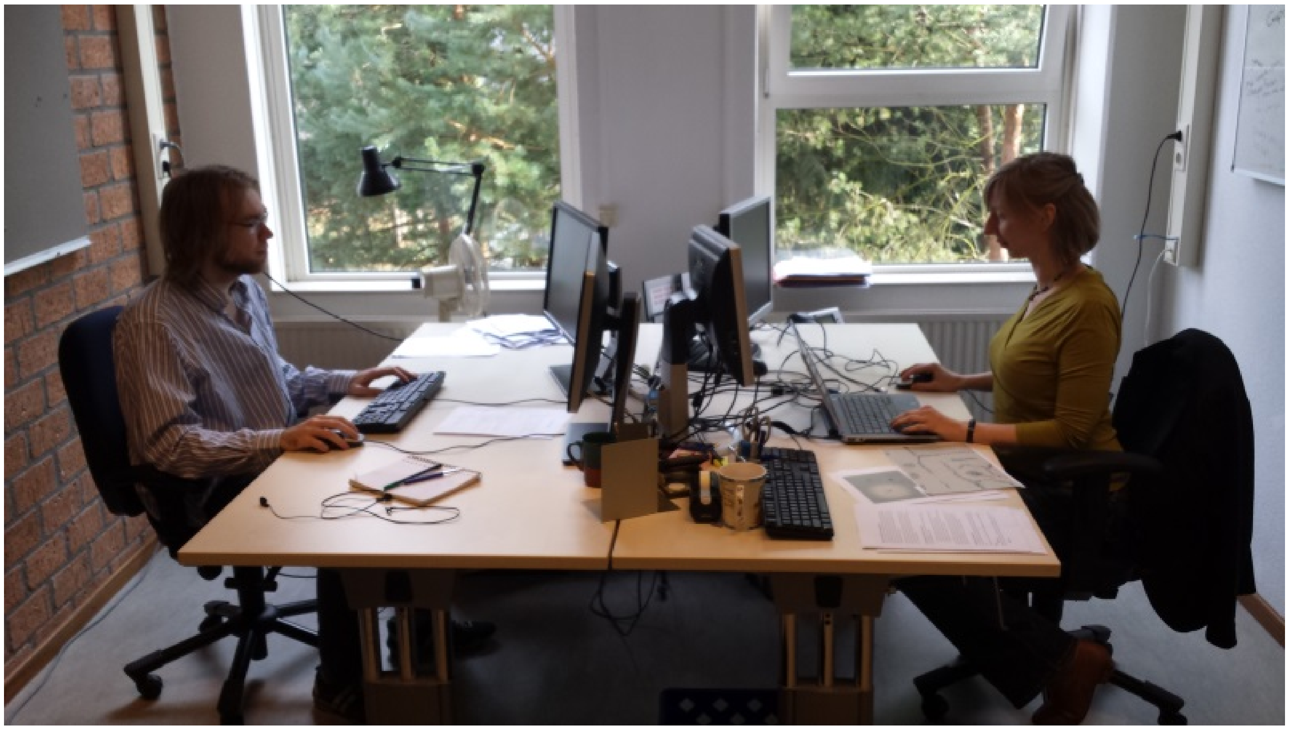


Figure 5.1: A picture of the experimental set-up.

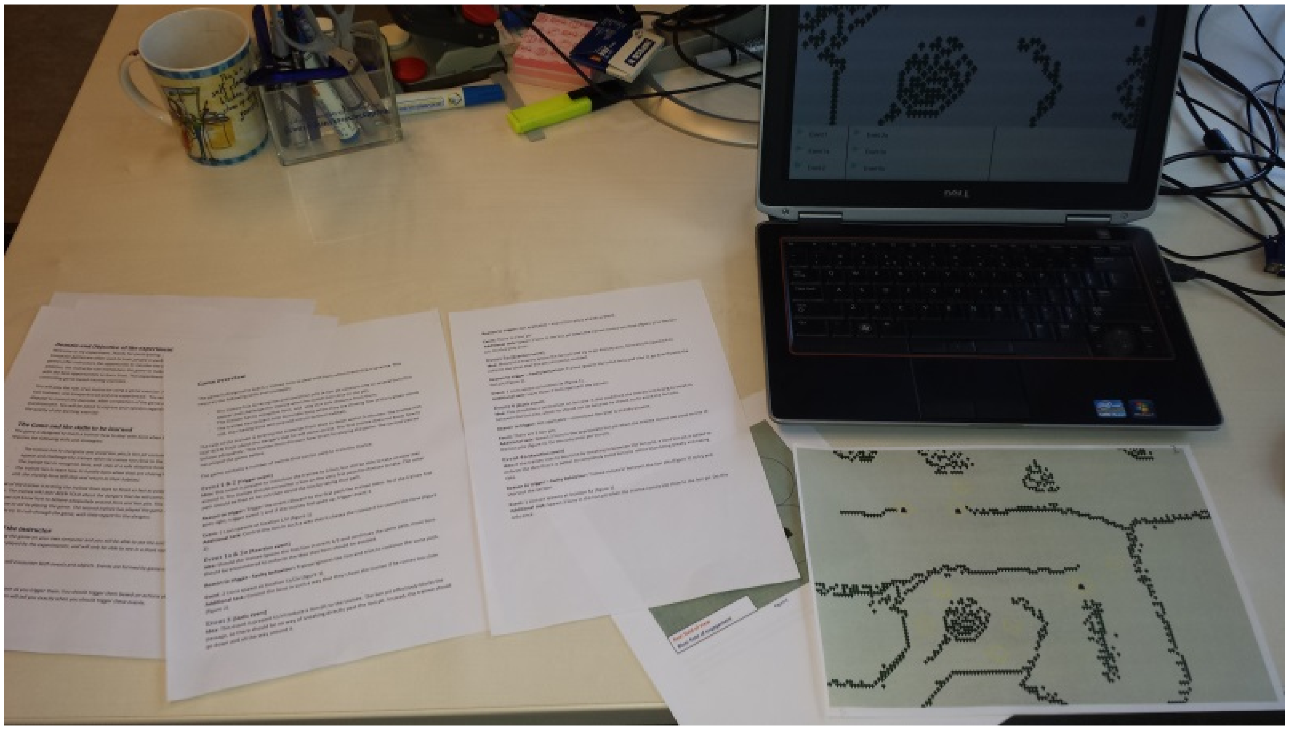


Figure 5.2: A picture of what the desk of the test subject would look like during the experiment with the unsupported version of the game.

The experiment consists of three parts: the introduction, in which the test subject is introduced to the role of instructor and to the game with the instructor interface. The training session part, in which the test subject plays the role of instructor in the game. And the questionnaire part, in which the test subject fills out the questionnaire.

5.4.1 Introduction

We first give the test subject a tutorial document (see appendix C for the tutorial document given to supported test subjects and appendix D for the tutorial document given to unsupported test subjects) that gives an overview of the concept of being an instructor, the goals and tasks for the test subject during the game, what events are and how to trigger them, and the game concept. Specifically, the instructor is told that he has to ensure that the trainee needs to reach the following learning goals:

- The trainee has to recognize and avoid lion pits (a lion pit contains one or several lions that appear and challenge the trainee when he comes too close to the pit).
- The trainee has to recognize lions, and stay at a safe distance from them.
- The trainee has to learn how to handle lions when they are chasing him (if the trainee stands still, the chasing lions will stop and return to their habitat).

No specific learning focus is introduced for either trainee. The instructor has to ensure that the trainee is trained in relationship to all of the learning goals. The instructor is therefore encouraged to give the trainee as much practice as possible when it comes to these learning goals, to inspire the instructor into adding custom events. Furthermore, the instructor is asked to try to keep the trainee alive throughout the scenario, to ensure that the instructor does not make it needlessly difficult with no regard to the trainee.

If the instructor is in the supported group, the document will also explain the automated A.I. elements that control the lions and lion pits. Conversely, the unsupported group has to manually ensure that the lions and lion pits show the correct behaviour. How to do this is explained in the document.

After reading through the tutorial document, the test subject does a practice session under guidance of the experiment leader to get familiar with the game and the respective instructor interface (with ISM for the supported group, without ISM for the unsupported group). The practice session is done in a special map separate from the map with which the training session is done.

5.4.2 Training session

Before the training session, the test subject is given an overview of the scenario in which he needs to be the instructor for a trainee (see appendix E for the scenario overview given to supported test subjects and appendix F for the scenario overview given to unsupported test subjects). The supported group gets to view the scenario in-game, with the use of the ISM interface. In addition, they are given a short document that provides some additional detail on the events. The unsupported group gets a picture of the in-game map and a document describing the events present within the scenario. Both groups gets exactly the same information.

After that, the training session is done twice with two different trainees, separated by a short optional pause to review the scenario again. The test subject is told that the first trainee is inexperienced, and that the second trainee is experienced. Both trainees are played by the experiment leader with the use of a script (see appendix H). The scripts detail the learning state of the trainee and how the trainee should react to every event, and every reaction event that the test subject adds or activates.

A sample taken from the script for the inexperienced trainee (brackets indicate an update in the learning state):

1. Go up at first junction.

2. When lion comes.
    (a) Run.
    (b) Let the lion hit the trainee character a few times (around 4 times).
    (c) Run into corner.
    (d) Cannot move.
    (e) Lion runs back. [**cautious of lions - the trainee now recognizes lions as a potential danger and will run away when he notices them**]
3. Go the other way.

5.4.3 Questionnaire

After the training session, the test subject is asked to fill in the questionnaire. After filling in the questionnaire, the experiment is over. For the questionnaire, please refer to appendix I.

6 | Results

We collected the results from both the in-game data collection and the questionnaire and grouped them into two groups: the supported group (those who played the game supported by ISM), and the unsupported group (those who played the unsupported game).

Only a small part of our results turned out to have a normal distribution. As a result, we used a non-parametric test to test our data. We used the median and the interquartile range to determine the results, and we used the Kruskal-Wallis test to describe the significance of the results. For the complete overview of all the results, please refer to appendix J.

6.1 Test subjects

A total of 17 test subjects participated in the test, of which 9 were in the supported group, and 8 in the unsupported group. The average age of a participant is 24 ($\sigma = 2.42$), with 11 identifying as male, and 6 as female.

6.2 Data types

The data collected from the experiment are grouped into two parts: the in-game data and the questionnaire data. The questionnaire data type is similar for all questions, every question has the test subject fill in a value of 1, 2, 3, 4, 5, or 0. Every value represents the level of agreement from the trainee, ranging from 5, strongly agreeing to the statement, to 1, strongly disagreeing with the statement. The value of 0 represents the "not applicable" option.

The in-game data types differs per data collection. Most data types are fairly straight forward, and are simply a count of actions occurring within the game:

1. Total events created.
2. Total events adapted.
3. Total events triggered.
4. Total number of lions present at the end of the scenario.
5. Total number of lion pits present at the end of the scenario.
6. Total number of left mouseclicks.
7. Total number of middle mouseclicks.
8. Total number of right mouseclicks.

Others count down from a maximum value:

1. Time left.

2. Trainee health left

Finally, there is a data type that shows which events were triggered during the training session, and is a collection of booleans:

1. Per event: whether it was triggered.

6.3 Hypotheses

- **Hypothesis 1**:
  With ISM the instructor will have a better overview of the scenario.

  There were two questions regarding the overview before the training session started in the questionnaire, one for the first training session, and one for the second training session. We used the results from the questionnaire for the first training session only, since it is the first time the test subject comes into contact with the scenario, since the same scenario is used for the second training session as well. The game overview (the overview the trainee has of the game: the user interface, controls, etc.) should also be improved or should at least stay the same. If the test subject has no good overview of the overall game, the overview of the scenario could suffer as a result. The question for the game overview was not specific to a session.

  Overview scenario - first trainee

	Median	Interquartile Range	Significance
Unsupported	3.5	1	0.118
Supported	4	2	

Table 6.1: Overview scenario - first trainee.

  Overview game

	Median	Interquartile Range	Significance
Unsupported	4	2	0.336
Supported	4	2	

Table 6.2: Overview game.

- **Hypothesis 2**:
  With ISM the instructor will plan more often before the scenario starts.

  The test subject is asked whether he had a plan before the session started at two points in time: one for each session.

  Had a plan before the session - first trainee

	Median	Interquartile Range	Significance
Unsupported	3	2	0.253
Supported	4	2	

Table 6.3: Had a plan before the session - first trainee.

  Had a plan before the session - second trainee

	Median	Interquartile Range	Significance
Unsupported	4	1	0.526
Supported	5	1	

Table 6.4: Had a plan before the session - second trainee.

- **Hypothesis 3**:
  With ISM the instructor will make adjustments more easily in the scenario. Therefore the instructor will make more adjustments on average.

  We only use the data collected for adding custom events, since adapting events is only possible in the supported version, so the results would be skewed.

  Custom events added - first trainee

	Median	Interquartile Range	Significance
Unsupported	0	1	0.245
Supported	1	2	

Table 6.5: Custom events added - first trainee.

Custom events added - second trainee

	Median	Interquartile Range	Significance
Unsupported	0.5	2	0.473
Supported	1	5	

Table 6.6: Custom events added - second trainee.

- **Hypothesis 4**:
  ISM will result in the challenge level of the scenario being more appropriate for the trainee.

  The challenge level is reflected in the hit points and the time left at the end of the scenario. Low hit points indicate that the trainee was hit by a lot of lions and therefore had a challenging scenario. The time left reflects the same, if the trainee had to handle a lot of lions and lion pits it will cost time and therefore he had a challenging scenario. The hit points start at 20 and can go down to 0 in increments of 1. The time starts at 300 and can go down to 0 in increments of 1.

  Hit points left - first trainee

	Median	Interquartile Range	Significance
Unsupported	9.5	11	0.146
Supported	7	3	

Table 6.7: Hit points left - first trainee.

Time left - first trainee

	Median	Interquartile Range	Significance
Unsupported	139	47	0.014
Supported	109	7	

Table 6.8: Time left - first trainee.

Hit points left - second trainee

	Median	Interquartile Range	Significance
Unsupported	11	6	0.020
Supported	5	8	

Table 6.9: Hit points left - second trainee.

Time left - second trainee

	Median	Interquartile Range	Significance
Unsupported	171	42	0.101
Supported	153	16	

Table 6.10: Time left - second trainee.

- **Hypothesis 5**:
  ISM will be accepted by the instructor because it is usable and because it is technologically sound.

  We use the data collected from the questionnaire regarding whether the game showed the behaviour expected and whether the interface was easy to use.

  The game showed expected behaviour

	Median	Interquartile Range	Significance
Unsupported	4.5	2	0.719
Supported	4	3	

Table 6.11: The game showed expected behaviour.

The interface was easy to use

	Median	Interquartile Range	Significance
Unsupported	3	2	0.100
Supported	4	2	

Table 6.12: The interface was easy to use.

- **Hypothesis 6**:
  ISM results in the instructor having a higher opinion of his own performance in triggering, adapting, and adding events.

  We use the data collected from the questionnaire regarding whether the test subject felt like he could trigger, and add events correctly and in a timely manner.

  Was able to correctly and timely trigger events

	Median	Interquartile Range	Significance
Unsupported	3	2	0.487
Supported	4	2	

Table 6.13: Was able to correctly and timely trigger events.

Was able to correctly and timely add events

	Median	Interquartile Range	Significance
Unsupported	3.5	2	0.483
Supported	4	2	

Table 6.14: Was able to correctly and timely add events.

7 | Discussion

In this chapter we will discuss the results of the experiment. We will start off by discussing what the results mean for our hypotheses. We will then conclude with some additional observations.

Hypothesis 1:
With ISM the instructor will have a better overview of the scenario.

The result of test subject's overview of the scenario seems to show some trends pointing towards ISM improving the overview of the scenario before starting the scenario (see table 6.1).

The result of the test subject's overview of the game is too insignificant to give any justifiable indications for ISM (see table 6.2). The similar medians combined with the high interquartile range suggests that the answer mostly depends on the test subject rather than whether the game was supported. The impact of personality coupled with our relatively small group of test subjects most likely translates into the wide variety in this result between test subjects.

Hypothesis 2:
With ISM the instructor will plan more often before the scenario starts.

The result of test subject's plan for the first trainee is too insignificant to draw any conclusions regarding the hypothesis (see table 6.3). This insignificance most likely stems from the fact that our measurements here were insufficient, since the test subject has to reflect back on whether he had made a plan before each session. Unfortunately, the test subject might not correctly recall whether they had made a plan, resulting in a wide variety between test subjects. Furthermore, the test subjects were unfamiliar with the game and the surrounding domain. As such, some test subjects might plan everything ahead, while others might just want to start the scenario and see what happens. Which makes the results heavily dependent on the personalities of the test subjects. This is noticeable in the interquartile range, as the interquartile range is relatively high for both supported and unsupported test subjects.

The result of test subject's plan for the second trainee is also too insignificant to draw any conclusions regarding the hypothesis (see table 6.4). The reason behind this insignificance is most likely similar to the reason that we discussed in the previous paragraph. We conclude that it is likely that whether they planned depends more on the test subject's own personality rather than on the support ISM provides. On a separate note, from the results it appears as if test subjects planned more for the second trainee than for the first trainee. This is most likely due to the fact that they are more familiar with the scenario by the time they do the session with the second trainee. We confirmed this by doing a Wilcoxon signed-rank test comparing whether the test subject had a plan for the first trainee with whether the test subject had a plan for the second trainee (see table 7.1). From this result, it can indeed by concluded that the test subjects planned more for the second trainee on average, regardless of whether they were supported or unsupported.

Had a plan before the session, difference between first and second trainee

	N	Mean Rank	Sum of Ranks	Significance
Negative Ranks[1]	1	9.50	9.50	0.017
Positive Ranks[2]	11	6.23	68.50	
Ties[3]	5			
Total	17			

Table 7.1: Had a plan before the session, difference between first and second trainee.

Hypothesis 3:
With ISM the instructor will make adjustments more easily in the scenario. Therefore the instructor will make more adjustments on average.

The lack of the significance in both of the results of the amount of events added by the instructor (see table 6.5 and table 6.6) are likely explained by a curious phenomenon we observed during the experiment. Either a test subject would add little to no events whatsoever, or add a lot of events. We observed no test subject who held a middle ground. Moreover, test subjects who added a lot of events seemed more likely to ignore the events already present on the map. All test subjects held back on adding events in the first scenario, opting to follow the scenario as written. The test subjects who did add events on their own seemed to do so in the second scenario. There are no trends regarding ISM to be observed here.

Hypothesis 4:
ISM will result in the challenge level of the scenario being more appropriate for the trainee.

The result of the amount of hit points the trainee had left at the end of the first scenario shows that the trainee actually had less hit points (i.e. a more difficult time) at the end of the first scenario when the instructor was supported by ISM, compared to an unsupported instructor (see table 6.7). However, the interquartile range suggests that ISM might help the instructor in providing the correct difficulty level. Specifically, certain test subjects did not seem proficient. This is due to the fact that they had difficulty controlling the game. I.e. they were slow at triggering, creating, and controlling events. This resulted in the trainee being able to easily pass most events without difficulty. In one instance of the experiment, the trainee was able to pass by every event without it being triggered in time by the instructor. This explains the difference between both the medians and the interquartile ranges: proficient test subjects do well in adjusting the challenge level regardless of being supported, whereas test subjects who are not proficient have difficulty when not supported by ISM.

This explanation also translates into the result of the amount of time the trainee had left at the end of the scenario, and this result has significance. Again, both the median and interquartile ranges for the time left for the trainee at the end of the first scenario are less for test subjects that were supported by ISM (see table 6.8). Similar to the first result, a lower result means that the trainee had a more difficult time. We argue that the explanation for the first results holds for this result as well.

The result of the amount of hit points the trainee had left at the end of the second scenario is significant and shows us that the trainee had less hit points at the end of the second scenario for the ISM supported version (see table 6.9). Interestingly, the interquartile range for the supported version seems to have increased greatly, while the interquartile range for the unsupported version has decreased. This is probably due to two phenomena: first, the fact that the second trainee is much faster than the first one, and actively tries to rush past things. As such, most test subjects playing the unsupported version are likely to be late in triggering events. Second, since the second trainee is

[1]The test subject had more planned for the first trainee than for the second trainee.
[2]The test subject had more planned for the second trainee than for the first trainee.
[3]The test subject had equal amount planned for both trainees.

experienced, he knows how to handle the events, as such, properly triggered events do not necessarily translate in lost hit points. Instead, if the test subject is late in triggering events, it may very well happen that the trainee is suddenly surrounded by lions. Since the second trainee is fast, test subjects have made these mistakes. Summarizing: if the test subject was unsupported, he was more likely too late to trigger an event, resulting in the trainee rushing past the event and losing little to no hit points. Most test subjects were late, translating in a high median and a low interquartile range. Conversely, if the test subject was supported, it is possible that he is slightly late with an event, resulting in the trainee losing more hit points because he is suddenly in danger (e.g. lions spawning next to him). Only some test subjects were late, translating in a low median and a high interquartile range.

The result of the amount of time the trainee had left at the end of the second scenario suggests a trend that corresponds to the previously discussed result. The time left for the trainee at the end of the second scenario is lower for the supported trainee (see table6.10). The fact that the second trainee, who is experienced, is more likely to not lose hit points from events does not translate into the trainee not being delayed by the event. As such, this trend suggest that test subjects supported by ISM were able to provide more of a challenge for the second trainee, since the median is lower. The interquartile range suggests that the test subjects were able to provide close to the same level of challenge for this trainee. The interquartile range for the unsupported version is likely due to the fact that test subjects who are proficient at the game were able to provide a good level of challenge, whereas players who are not proficient at the game were not.

Hypothesis 5:
ISM will be accepted by the instructor because it is usable and because it is technologically sound.

The result of the expected game behaviour is insignificant and provides us with no insight whether ISM provides the behaviour expected by the test subject (see table 6.11). It is likely that the disparity between the test subjects who were proficient and the test subjects who were not translates into a similar disparity between results. Specifically, proficient test subjects are more likely to have certain expectations of the technology, and therefore be more critical of ISM as a whole. Whereas test subjects who are not proficient are less likely to pay attention to how the technology behaves, since they are more likely to concentrate on executing the scenario, which results in them giving an average or higher grade, simply because the technology did not show any obvious problems.

The result of the ease of use of the interface shows a trend suggesting that ISM has an interface that is easy to use, especially compared to the rudimentary interface provided by the unsupported version (see table 6.12).

Hypothesis 6:
ISM results in the instructor having a higher opinion of his own performance in triggering, adapting, and adding events.

Both results of the test subject's ability to correctly and timely trigger events, and the test subject's ability to correctly and timely add events is insignificant and does not allow us to draw a proper conclusion concerning ISM (see table 6.14 and table 6.13). The relatively high interquartile range in both results shows that there is a relevant difference between all people. This is most likely due to the disparity in game proficiency between the test subjects. A proficient test subject will likely have an easier time correctly and timely triggering events, and correctly and timely adding events. Conversely, a test subject who is not proficient will likely have a harder time.

Moreover, the relative simplicity of the game (especially compared to VBS2) means that the test subject is more likely to have a high opinion of his own performance, whether he was supported or not.

7.1 Impact of game proficiency

In our discussion of the hypotheses we frequently hypothesize that the game proficiency of the test subjects largely impacts the results. Specifically, test subjects who seemed proficient in playing the game (i.e. had no problem controlling individual units with the mouse) seemed to have less to no trouble controlling the units in the unsupported game. Additionally, proficient players seemed to grow bored more easily in the supported version, because they did not have much to do (compared to a normal game).

The observation of proficiency is most present in the results whether the challenge level for the trainee was more appropriate (fourth hypothesis), where we introduce the phenomenon. Moreover, the observation of proficiency is reinforced by the test subject's personal preference theorized in the result of the test subject's overview of the game (first hypothesis). As this personal preference may equate to the proficiency of the test subject. Specifically, test subjects who are proficient at the game will likely have a good overview of the game, whereas test subjects who are not proficient at the game are less likely to have a good overview of the game. Moreover, our observations in the results of the amount of events added by the instructor (third hypothesis) could also be linked to the proficiency of the test subject. It is likely that proficient test subjects are more likely to add events on their own since they are more confident in controlling the game, and by extension, adding events. Conversely, test subjects who are not proficient are less likely to be confident in controlling the game, and therefore more likely to stick to the scenario exactly as it is written. The observation of proficiency is also present in the acceptance of ISM (fifth hypothesis), as proficient test subjects are likely to be more critical of ISM, whereas test subjects who are not proficient are likely to be satisfied with the technology if it does not present any glaring problems. Finally, the observation of proficiency is likely present in the test subject's opinion of his own performance (sixth hypothesis), where test subjects who are proficient are more likely to have a high opinion of themselves than test subjects who are not proficient, since proficient test subjects are confident and will have an easier time controlling the game.

We speculate that the proficiency of the players correlates to their proficiency with Real-Time Strategy (RTS) games, due to the fact that it shares a similar control scheme (mouse to move units), a similar point of view (top-down), and that controlling multiple units in a real-time environment (time progresses even if you do nothing) is a key component of RTS games. Unfortunately, we did not ask the test subjects about their proficiency in RTS games.

8 | Conclusion and future work

Based on the results we have gathered and discussed, we argue that ISM probably supports the instructor in a variety of ways. However, we believe that our implementation of the model, i.e. the game that we developed, was too simple and that our group of test subjects was too varied in proficiency to show significant results.

Our results show trends that leads us to believe that ISM supports the instructor in providing the optimal learning experience for the trainee. The results of our first hypothesis does show a trend that suggests that the instructor would have a better overview of the scenario when presented with the in-game scenario view ISM provides. Furthermore, the results of our fourth hypothesis suggests that ISM helps the instructor in providing an appropriate level of challenge for the trainee. Finally, the interface used in our implementation seems to be accepted by the test subjects.

Unfortunately, the wide variety between test subjects, caused by what we believe to be the proficiency of the test subject shrouds most of our results. We observed this disparity in proficiency during our experiment, where certain people would have a much easier time using the game than others. We argue that the proficiency of the test subject directly influences the test subject's overview of the game, whether the test subject has a plan, whether the test subjects makes adjustments to the scenario, the resulting challenge level of the scenario, the expected game behaviour, and the test subject's opinion of his own performance. As a result of this vast influence, we cannot confidently give a direct answer to our hypotheses.

Our test subject group was both too small and too varied to extract significant results. In order to appropriately test ISM, the test subjects would all have to be proficient with RTS games, or alternatively, the test subjects are all not proficient with RTS games. Another possibility would be to have a bigger test subject group, record their proficiency with RTS games through the use of the questionnaire, and take this data into account as covariate while doing the analyses. We propose that the test subject's proficiency could be tested with the use of three questions:

- How many hours per week do you spend playing computer games?
- Do you know what a Real-Time Strategy (RTS) game is?
- How many hours per week do you spend playing RTS games?

The game is not complex enough to compare to VBS2, and as a result, not complex enough to fully show the possibilities ISM provides. In VBS2, many objects and units are available for the instructor to add, whereas our implementation only offers two types of objects. As a result, the user interface that is provided by ISM does not offer much improvement when trying to add a new event, because there is little improvement to be gained in such a simple implementation. Moreover, the limited scope of the game also means that is easier to understand and therefore use, even without the support provided by ISM. Test subjects that we believed to be proficient seemed to have little difficulty with the game, regardless whether they were supported. If the proficient test subjects were presented with a more complex game, we believe that they would benefit from ISM.

Despite our limited results, we believe that ISM has potential to provide ample support for the instructor. We recommend that further testing of ISM be done with a more complex game and a much larger test subject group, appropriately grouped up according to their proficiency. Alternatively, ISM could also be implemented in VBS2 and tested on a test subject group consisting of domain experts. This secondary option would be more focused on testing the practicality of ISM.

9 Acknowledgements

I would like to thank a great number of people who helped me in various ways during the course of this project. First of all, I would like to thank my supervisors, John-Jules Meyer and Annerieke Heuvelink for their great insight, extensive feedback, and general support. Moreover, I would like to thank Philip Kerbusch, Marieke Peeters, and Karel van den Bosch for their contribution and support in their area of expertise.

I would also like to thank all my test subjects, thanks for participating in my experiment. I promised you anonymity, so I will not mention your name, but you know who you are, and you are awesome!

I would like to thank my TNO room mates, Christian, Sylvia, and Pauline, for the insightful conversations, action-packed table tennis tournaments, and awesome board game nights.

Furthermore, I would like to thank all of my friends for their support and interest during the course of this project. Specifically, I would like to thank Christian, Michiel, Vivian, Jonas, Paul, and Marianne, for the epic dungeons and dragons nights, awesome board game nights (exception: Settlers of Catan), and wonderful film nights. Could have done it without you guys, but it would have been harder. ;)

Last, but most definitely not least, I would like to thank my parents for their continued support and love, not only during my project, but during my entire life. I would also like to thank my bird, Itsu, though she will probably never read this.

Bibliography

Guillaume Chanel, Cyril Rebetez, Mireille Bétrancourt, and Thierry Pun. Boredom, engagement and anxiety as indicators for adaptation to difficulty in games. In *Proceedings of the 12th international conference on Entertainment and media in the ubiquitous era*, pages 13–17. ACM, 2008.

Jenova Chen. Flow in games (and everything else). *Communications of the ACM*, 50(4):31–34, 2007.

Mihaly Csikszentmihalyi. *Flow: the psychology of optimal experience.* New York: Harper and Row, 1990.

Bohemia Interactive. Bohemia interactive simulations, a. URL `http://products.bisimulations.com`.

Bohemia Interactive. Virtual battlespace 2 simulation, b. URL `http://products.bisimulations.com/simulation`.

Jesper Juul. Fear of failing? the many meanings of difficulty in video games. *The video game theory reader*, 2:237–252, 2009.

Paul A. Kirschner, John Sweller, and Richard E. Clark. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2):75–86, 2006.

Thomas W. Malone. What makes things fun to learn? heuristics for designing instructional computer games. In *Proceedings of the 3rd ACM SIGSMALL symposium and the first SIGPC symposium on Small systems*, pages 162–169. ACM, 1980.

Thomas W. Malone. Toward a theory of intrinsically motivating instruction. *Cognitive Science*, 5(4): 333–369, 1981.

Thomas W Malone and Mark R Lepper. Making learning fun: A taxonomy of intrinsic motivations for learning. *Aptitude, learning, and instruction*, 3:223–253, 1987.

Marieke M. M. Peeters. Personalized educational games - developing agent-supported scenario-based training. 2014.

Hua Qin, Pei-Luen Patrick Rau, and Gavriel Salvendy. Effects of different scenarios of game difficulty on player immersion. *Interacting with Computers*, 22(3):230–239, 2010.

rude guss. Awesome 2d rts engine, 2009. URL `http://gmc.yoyogames.com/index.php?showtopic=422198`.

Penelope Sweetser and Peta Wyeth. Gameflow: a model for evaluating player enjoyment in games. *Computers in Entertainment (CIE)*, 3(3):3–3, 2005.

A | Game Engine Research

Points of research per game engine

1. **The complexity of the engine in general, whether the GUI is easy to use and how much programming is necessary to achieve a relatively simple game.**
2. **The programming language, whether there is someone proficient within the "Vergroten Toegankelijkheid van Serious Games" project group (or within TNO) with that language.**
3. **The community (userbase), whether it is active, extensive, and mature. The official support provided by the developer or through official channels.**
4. **What kind of extension modules are provided by the community or via the official support channels, and the robustness of these modules.**
5. **What kind of game bases (basic foundation for a certain type of game) are provided by the community or via the official support channels, and the robustness and relevance of these game bases.**
6. **The Integrated Development Environments (IDE) that are available for the game engine, and whether we are familiar with one of these environments.**
7. **The graphical possibilities of the engine, whether it can support 2 dimensional (2D) and/or 3 dimensional (3D). 2D games are easier to program and are also easier to make graphics for in the form of sprites.**
8. **LAN multiplayer capabilities.**
9. **Second screen capabilities.**
10. **Game engine performance, whether testimonials point out that the engine has performance issues.**
11. **Cost of the game engine, whether a free version is available, and the limitations of this free version.**

GameMaker

1. Relatively simple, not too complex. Probably the easiest to start in. GUI/interface should be no problem, since it is supported by the drag & drop interface.
2. Most work is done in the GUI (drag & drop), more advanced work is done in its own language: GML (Game Maker Language).
3. User-base / support is available. Good wiki and support forum.
4. User-made supporting libraries (DLL, Action), looks reasonably limited in terms of quantity.
5. Game bases are available but user-made, not robust.

6. Development environment in own GUI, because it is mostly drag & drop.
7. Engine is mostly 2D, 3D would be possible but the implementation would be more complex than a regular 3D engine.
8. Multiplayer is basic and unrefined: winsock-like.
    - Online adjustments possible, but it would probably take more work than other engines.
9. Second screen available through split screen. Tutorials on how to implement this are available.
10. Relatively poor performance, but that does not need to matter, since it probably should not have to support advanced graphics/physics.
11. Free version limited (limited resources).

Conclusion:
Easily accessible base, complexities such as 3D, multiplayer and possibly others could cause problems, but we are unsure at this point. If we want to make the game 2D and really simple then this is probably the best choice, but the undefined complexities are risky.

Unreal Engine 3 - Unreal Development Kit (UDK)

1. Pretty complex, with the use of the wealth of tutorials and support this is probably surmountable.
2. Own programming language: UnrealScript (object oriented language).
3. There is an enormous amount of information and support available. Strong wiki and user base. Unofficial sites are available with detailed tutorials (www.hourences.com).
4. Professional supporting libraries are available by default: Kismet, Bink Video Codec, SpeedTree, and more. Not all are relevant to us. User-made libraries are also available.
5. Game bases (starter kits) available on the wiki for certain types of games, it could be easy to base the conceptual game on one of these kits.
6. There are a bunch of development environments available for UnrealScript: WOTgreal, nFringe, and a couple of others. There is also a UDK plugin for Eclipse.
    - Kismet is a visual scripting system that can be used to give form to menus and scenarios in a visual manner. This could be very handy because it is as flexible as it looks, we could even write a scenario in this tool, which would make it much easier and clearer.
    - It is also possible to use Scaleform to create the UI, but this is much more complex and flexible, which is unnecessary for this project.
7. The engine is mostly in 3D, 2D would be possible but the implementation would be more complex than a regular 2D engine.
8. Network component is available. It supports automatically assigning roles to players, each role with their own rights (which allows us to easily separate the Instructor from the Trainee). No peer to peer.
9. Second screen available through split screen. Tutorials on how to implement this are available.
10. Performance seems fine.

11. There is a non-commercial license available.

Conclusion:
Flexible and powerful engine, has interesting tools such as Kismet. Robust starter kits could help us start on our game. The only problem we can find is the complexity of the engine and the "cooking" of maps, which can cause problems in the construction of the scenario environment (www.hourences.com/an-entire-simple-udk-game/#cooking).

Unity

1. Looks reasonably complex, but less complex than UDK. With the help of the tutorials and support we can probably overcome the complexity.
2. Programming languages: C#, Unity Script, or Boo.
3. Very strong user-base: the most active forum of all engines. Wiki is available but the documentation on the site is better.
4. Libraries are only supported in the pro version. An enormity of extensions and content is available via the Asset Store, but this usually costs money (Kismet is available in the Asset Store).
5. Game bases (starter kits) are available, but are somewhat less robust than the starter kits for UDK.
6. Unity provides its own editor as programming environment. It is possible to code or script using Unity.
7. The engine is mostly in 3D, 2D would be possible but the implementation would be more complex than a regular 2D engine.
8. Network component is available. Works via Remote Procedure Calls. No peer to peer.
9. Second screen available through split screen. Tutorials on how to implement this are available.
10. Performance seems fine.
11. The free version also introduces some additional restrictions compared to the pro version that will probably not hinder our project.

Conclusion:
Flexible and powerful engine. The extensions usually cost money, but this does not have to be a problem since we might not need extensions. It is less complex than UDK but it lacks robust starter kits. The strong community does provide a lot of components (e.g. http://forum.unity3d.com/threads/167856-PlayEditor).

VBS2

1. Specifically made for training and simulations, as such, complexity is relatively okay, we would probably have to write a plugin to handle abstract events.

2. A plugin can be written in the Application Scripting Interface (ASI, own language) or VBS2Fusion (C++, better performance).
3. Weak user-base, we were unable to find a good forum. Most information seems to originate from the wiki and this information is not very extensive. The knowledge and expertise from within TNO might be able to provide us with additional support.
4. Very little plugins and libraries. Only two on the official wiki.
5. Game bases are not really available, as it is more making an extension on the game. As such, the engine functions as a game base.
6. Virtually anything can be used for the development environment. The scripts can be written in any text editor, but there are no real editor plugins supporting this (e.g. for Eclipse). There are syntax definitions available for text editors such as Notepad++. The C++ API (VBS2Fusion) based plugin can be written in an IDE that supports C++.
7. The engine is in 3D, 2D would be nearly impossible.
8. Network component is already completely implemented in VBS2. Most of the work will go into the creation of the scenarios and the development of the supporting plugin.
9. Split screen does not seem to be possible in VBS2. A second screen solution seems to be hard to accomplish.
10. Performance seems fine, though A.I. is lacking (information gathered from research done within TNO).
11. License already exists within TNO.

Conclusion:
A different solution than using the other game engines. This is more of a ready to go simulation in which there is a lot of adaptation possible. It could save us a lot of time if the plugin system is good enough. VBS2 offers less possibilities than the other engines but probably fits well in our scope. The risk mostly lies at the power of the plugin system and the limited community.

Source Engine (Source SDK)

1. Complexity comparable with Unity, maybe a little less complex, since it more of a modding tool.
2. Programming language: C++ (languages such as LUA are available).
3. User-base / support reasonably available. Strong wiki but a mediocre support forum.
4. There are not a lot of plugins available, only a couple of third party applications.
5. Game bases are not really available.
6. Development environments that support C++ support the source SDK. Visual Studio comes recommended.
7. The engine is in 3D, 2D would be nearly impossible.
8. Available network component. Works via Game events. No peer to peer. Server has to implemented as a server plugin.

9. Second screen available through split screen. Tutorials on how to implement this are not strongly available, but it should be possible (forums.steampowered.com/forums/showthread.php?t=1857198).
10. Performance seems fine.
11. Source SDK is essentially free.

Conclusion:
We have our doubts with the use of this engine. The advantages of other engines are not available here but it is still pretty complex. All other components are available, however.

Off-line map adaptations probably have to be done through the map editor in the game engines themselves. On-line scenario adaptations can probably be achieved through the use of actions within the game from the instructors point of view. Saving these on-line adaptations can cause difficulty in certain engines through complexities such as “cooking”.

Proposal

If we want to make a 2D and topdown game, then GameMaker is the most likely candidate.

Unreal & Unity offer the most complexity, but this also requires more work. Both have our personal preference. Source also falls into this category, but I feel that both Unreal and Unity are better than the Source engine when it comes to developing a conceptual game.

VBS2 requires additional enquiries if it is possible to use it for what we want to achieve. We believe it is. We also believe it would fit our scope the best. It would require the least amount of work and time if it is possible to do everything we want to do in VBS2 (with the exception of GameMaker).

Therefore, if we want the game to be more complex than 2D and topdown, we should try Unreal or Unity. If this seems to be too complex, we should go over to VBS2.

B | Custom Field of View code

```
    if global.is_instructor && global.supported
    {
        for (i=0;i<32;i+=1)
        {
            for (z=0;z<18;z+=1)
            {
                if (fog[i,z] = true)
                {
                    if (dist[i,z] <= view + 75)
                    {
                        draw_set_alpha(((dist[i,z]-view)/view)/5);
                    }
                    else
                    {
                        draw_set_alpha(0.2);
                    }

                    draw_set_color(c_black);
                    //draw_sprite(fog,-1,view_xview[0]+(16 +
i*32),view_yview+(16 + z*32));

draw_rectangle(view_xview[0]+(i*32),view_yview[0]+(z*32),view_xview[0]+(i*32+32),view_y
                }
            }
        }
    }
```

C | Tutorial Document - Supported

Domain and Objective of the experiment

Welcome to my experiment. Thanks for participating. The experiment is on game-based training. Computer games are often used to train people in performing tasks and to learn new skills. Many games offer instructors the opportunity to monitor the behavior of the trainee in real time. In addition, the instructor can manipulate the game to make sure that the exercise provides the trainee with the best opportunities to learn from. This experiment investigates facilities to help an instructor controlling game-based training exercises.

You will play the role of an instructor using a game exercise. The experimenter will play the role of two trainees: one inexperienced and one experienced. You will have a number of facilities at your disposal to control the exercise. After completion of the game-exercise, you will receive a questionnaire. You will be asked to express your opinion regarding the quality of the facilities and the quality of the learning exercise.

The Game and the skills to be learned

The game is designed to teach a trainee how to deal with lions when traversing a savanna. This requires the following skills and strategies:

- The trainee has to recognize and avoid lion pits (a lion pit contains one or several lions that appear and challenge the trainee when he comes too close to the pit).
- The trainee has to recognize lions, and stay at a safe distance from them.
- The trainee has to learn how to handle lions when they are chasing him (if the trainee stands still, the chasing lions will stop and return to their habitat).

The task of the trainee is to bring the trainee from start to finish as fast as possible with a 5-minutes deadline. The trainee HAS NOT BEEN TOLD about the dangers that he will come across. The first trainee does not know how to behave adequately around lions and lion pits. This trainee must discover how to act by playing the game. The second trainee has played the game before. This trainee might try to rush through the game, with little regard for the dangers.

The task of the instructor

You will be playing the game on your own computer and you will be able to see the entire savannah. The trainee will be played by the experimenter, and will only be able to see in a short radius around him.

During the game, you will encounter both events and objects. Events are formed by game objects.

Events

- **Trigger events**

Trigger events happen as soon as you trigger them. You should trigger them based on actions of the trainee. The event description will tell you exactly when you should trigger these events.

- **Static events**

Static events are already present in the scenario. They will not have to be triggered.

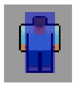
- **Reaction events**

Reaction events are linked to either a **static** or a **trigger event**. Reaction events happen as soon as you trigger them. You should trigger them based on reactions of the trainee to the linked event. The event description will tell you exactly when you should trigger these events.

- **New events**

By middle clicking anywhere on the map, it is possible to add an event at that position. Use this to make the scenario more difficult.

Objects

- **Trainee** 

Trainee has to make it from the start of the map (bottom left) to the finish at the end of the map (top right). There will be a health bar on top of the trainee, if this health bar reaches 0, the trainee will die.

- **Finish**

Finish has to be reached by the trainee in order for the scenario to successfully end.

- **Time**

Time will be displayed in the upper left corner of the screen. If it reaches 0, the trainee has failed to complete the scenario in time.

- **Lions**

Lions can attack the trainee and drain his health bar. They will attack the trainee if he comes too close and only move back when he is standing still.

- **Lion pits**

Lion pits are home to the lions. If the trainee comes too close to a lion pit, a number of lions will come out.

Events consist of either encountering lions or lion pits.

In this game, you perform the role of instructor. It is your task to monitor what the trainee is doing and, -if needed- take the actions in the exercise that make the trainee discover what is going on and what is the best way to perform the task. It is important that your actions do not cause the trainee's mission failure (e.g. it should not lead to killing the trainee).

Actions: You can take the following actions:

- trigger an event (circles on the map indicate potential events; you -as the instructor- can trigger the event if and when you think it is appropriate).
- control an event (e.g. controlling the behavior of the lions when they appear after an event is executed).
- create an event (e.g. if you consider it necessary, spawn one or more lions or lion pits at any position).

Evaluating instructional facilities

You are asked to perform the role of instructor in an interactive game-based exercise. In order to investigate the facilities that help an instructor controlling the exercise, we ask you to rate your experiences and give your opinions after the game. Some questions may be rated easily. Others may be more difficult. But try to rate these questions too as good as possible to the best of your impression. Please use all available points of the scale. Thus, if you think that a particular statement is right on the mark; then tick “strongly agree”. And, if you think a particular statement is in full contrast with your experience, do not hesitate to tick the other end of the scale: “strongly disagree”.

Do you have any questions or remarks?

D | Tutorial Document - Unsupported

Domain and Objective of the experiment

Welcome to my experiment. Thanks for participating. The experiment is on game-based training. Computer games are often used to train people in performing tasks and to learn new skills. Many games offer instructors the opportunity to monitor the behavior of the trainee in real time. In addition, the instructor can manipulate the game to make sure that the exercise provides the trainee with the best opportunities to learn from. This experiment investigates facilities to help an instructor controlling game-based training exercises.

You will play the role of an instructor using a game exercise. The experimenter will play the role of two trainees: one inexperienced and one experienced. You will have a number of facilities at your disposal to control the exercise. After completion of the game-exercise, you will receive a questionnaire. You will be asked to express your opinion regarding the quality of the facilities and the quality of the learning exercise.

The Game and the skills to be learned

The game is designed to teach a trainee how to deal with lions when traversing a savanna. This requires the following skills and strategies:

- The trainee has to recognize and avoid lion pits (a lion pit contains one or several lions that appear and challenge the trainee when he comes too close to the pit).
- The trainee has to recognize lions, and stay at a safe distance from them.
- The trainee has to learn how to handle lions when they are chasing him (if the trainee stands still, the chasing lions will stop and return to their habitat).

The task of the trainee is to bring the trainee from start to finish as fast as possible with a 5-minutes deadline. The trainee HAS NOT BEEN TOLD about the dangers that he will come across. The first trainee does not know how to behave adequately around lions and lion pits. This trainee must discover how to act by playing the game. The second trainee has played the game before. This trainee might try to rush through the game, with little regard for the dangers.

The task of the instructor

You will be playing the game on your own computer and you will be able to see the entire savannah. The trainee will be played by the experimenter, and will only be able to see in a short radius around him.

During the game, you will encounter both events and objects. Events are formed by game objects.

Events

- **Trigger events**

Trigger events happen as soon as you trigger them. You should trigger them based on actions of the trainee. The event description will tell you exactly when you should trigger these events.

- **Static events**

Static events are already present in the scenario. They will not have to be triggered, but you might still have to control the game elements related to a static event.

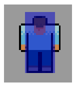
- **Reaction events**

Reaction events are linked to either a **static** or a **trigger event**. Reaction events happen as soon as you trigger them. You should trigger them based on reactions of the trainee to the linked event. The event description will tell you exactly when you should trigger these events.

- **New events**

By middle clicking anywhere on the map, it is possible to add an event at that position. Use this to make the scenario more difficult.

Objects

- **Trainee** 

Trainee has to make it from the start of the map (bottom left) to the finish at the end of the map (top right). There will be a health bar on top of the trainee, if this health bar reaches 0, the trainee will die.

- **Finish**

Finish has to be reached by the trainee in order for the scenario to successfully end.

- **Time**

Time will be displayed in the upper left corner of the screen. If it reaches 0, the trainee has failed to complete the scenario in time.

- **Lions**

Lions can attack the trainee and drain his health bar.

- **Lion pits**

Lion pits are home to the lions.

Events consist of either encountering lions or lion pits.

In this game, you perform the role of instructor. It is your task to monitor what the trainee is doing and, -if needed- take the actions in the exercise that make the trainee discover what is going on and what is the best way to perform the task. It is important that your actions do not cause the trainee's mission failure (e.g. it should not lead to killing the trainee).

Actions: You can take the following actions:

- trigger an event (circles on the map indicate potential events; you -as the instructor- can trigger the event if and when you think it is appropriate).
- control an event (e.g. controlling the behavior of the lions when they appear after an event is executed).
- create an event (e.g. if you consider it necessary, spawn one or more lions or lion pits at any position).

Instructional strategy: you should adhere to the following instructional strategies:

- if the trainee comes within sight of a lion, make the lion approach the trainee. As soon as the trainee stands still, direct the lions to a distant position (out of sight).
- If the trainee comes within sight of a lion pit, spawn one or more lions and make them approach the trainee. Once the trainee stands still, direct the lions to a distant position (out of sight).

Evaluating instructional facilities

You are asked to perform the role of instructor in an interactive game-based exercise. In order to investigate the facilities that help an instructor controlling the exercise, we ask you to rate your experiences and give your opinions after the game. Some questions may be rated easily. Others may be more difficult. But try to rate these questions too as good as possible to the best of your impression. Please use all available points of the scale. Thus, if you think that a particular statement is right on the mark; then tick “strongly agree”. And, if you think a particular statement is in full contrast with your experience, do not hesitate to tick the other end of the scale: “strongly disagree”.

Do you have any questions or remarks?

E | Scenario overview - Supported

Event 1 & 2 **[Trigger event]**

Idea: This event is present to introduce the trainee to a lion, but still be able to take an easy way around it. The trainee should encounter a lion on the very first path he chooses to take. The other path should be free so he can then avoid the lion by taking that path.

Event 1a & 2a **[Reaction event]**

Idea: Should the trainee ignore the first lion in event 1/2 and continues the same path, more lions should be encountered to enforce the idea that lions should be avoided.

Event 3 **[Static event]**

Idea: This event is present to introduce a lion pit to the trainee. The lion pit effectively blocks the passage, so there should be no way of sneaking directly past the lion pit. Instead, the trainee should go down and all the way around it.

Event 3a **[Reaction event]**

Idea: Should the trainee ignore the lion pit and try to go directly past, lions should approach to enforce the ideas that lion pits should be avoided.

Event 4 **[Static event]**

Idea: This should be a second test on lion pits. It also could lure the trainee into trying to sneak in between the lion pits, which he should not do because he should try to avoid the lion pits.

Event 4a **[Reaction event]**

Idea: If the trainee tries to buy time by sneaking in between the lion pits, a third lion pit is added to enforce the idea that it is better to completely avoid lion pits rather than being sneaky and taking risks.

F | Scenario overview - Unsupported

Game overview

The game is designed to teach a trainee how to deal with lions when traversing a savanna. This requires the following skills and strategies:

- The trainee has to recognize and avoid lion pits (a lion pit contains one or several lions that appear and challenge the trainee when he comes too close to the pit).
- The trainee has to recognize lions, and stay at a safe distance from them.
- The trainee has to learn how to handle lions when they are chasing him (if the trainee stands still, the chasing lions will stop and return to their habitat).

The task of the trainee is to bring the traverser from start to finish within 5-minutes. The trainee HAS NOT BEEN TOLD about the dangers that he will come across. The first trainee does not know how to behave adequately. This trainee must discover how to act by playing the game. The second trainee has played the game before.

The game contains a number of events that can be used to train the trainee.

Event 1 & 2 **[Trigger event]**

Idea: This event is present to introduce the trainee to a lion, but still be able to take an easy way around it. The trainee should encounter a lion on the very first path he chooses to take. The other path should be free so he can then avoid the lion by taking that path.

Reason to trigger: Trigger the event relevant to the first path the trainee takes. So if the trainee first goes right, trigger event 1 and if the trainee first goes up, trigger event 2.

Event: 1 Lion spawns at location 1/2 (figure 1).
Additional task: Control the lion in such a way that it chases the trainee if he comes too close (figure 2).

Event 1a & 2a **[Reaction event]**

Idea: Should the trainee ignore the first lion in event 1/2 and continues the same path, more lions should be encountered to enforce the idea that lions should be avoided.

Reason to trigger - faulty behaviour: Trainee ignores the lion and tries to continue the same path.

Event: 2 Lions spawn at location 1a/2a (figure 1).
Additional task: Control the lions in such a way that they chase the trainee if he comes too close (figure 2).

Event 3 **[Static event]**

Idea: This event is present to introduce a lion pit to the trainee. The lion pit effectively blocks the passage, so there should be no way of sneaking directly past the lion pit. Instead, the trainee should go down and all the way around it.

For figure 1, please see appendix G

Reason to trigger: Not applicable – event (lion pit) is already present.

Event: There is a lion pit.
Additional task: Spawn 2 lions in the lion pit when the trainee comes too close (figure 2) to the lion pit. Do this only once.

Event 3a **[Reaction event]**

Idea: Should the trainee ignore the lion pit and try to go directly past, lions should approach to enforce the ideas that lion pits should be avoided.

Reason to trigger – faulty behaviour: Trainee ignores the initial lions and tries to go directly past the lion pit (figure 1).

Event: 3 Lions spawn at location 3a (figure 1) .
Additional task: Have these 3 lions approach the trainee.

Event 4 **[Static event]**

Idea: This should be a second test on lion pits. It also could lure the trainee into trying to sneak in between the lion pits, which he should not do because he should try to avoid the lion pits.

Reason to trigger: Not applicable – event (two lion pits) is already present.

Event: There are 2 lion pits.
Additional task: Spawn 2 lions in the appropriate lion pit when the trainee comes too close to one of the lion pits (figure 2). Do this only once per lion pit.

Event 4a **[Reaction event]**

Idea: If the trainee tries to buy time by sneaking in between the lion pits, a third lion pit is added to enforce the idea that it is better to completely avoid lion pits rather than being sneaky and taking risks.

Reason to trigger – faulty behaviour: Trainee moves in between the lion pits (figure 1) to try and shortcut the section.

Event: 1 Lion pit spawns at location 4a (figure 1).
Additional task: Spawn 3 lions in the lion pit when the trainee comes too close to the lion pit. Do this only once.

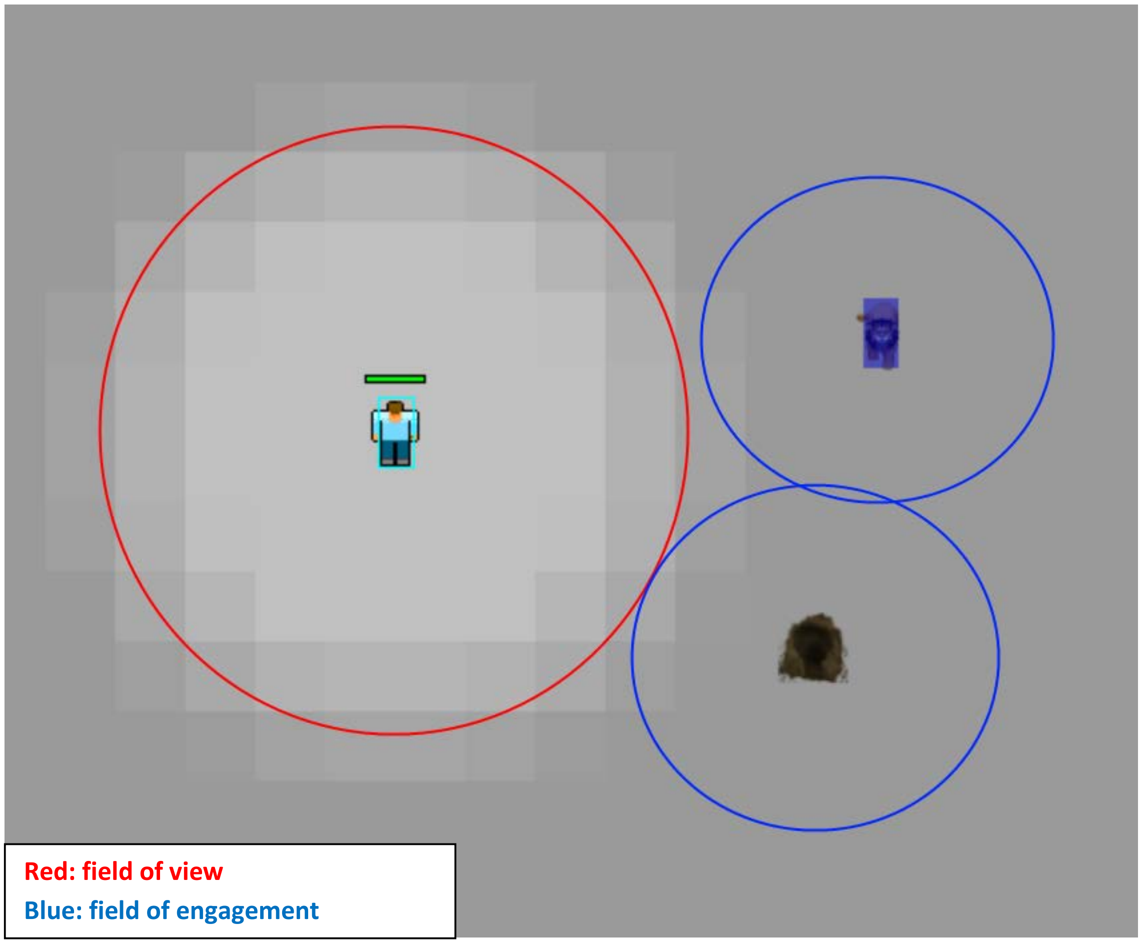
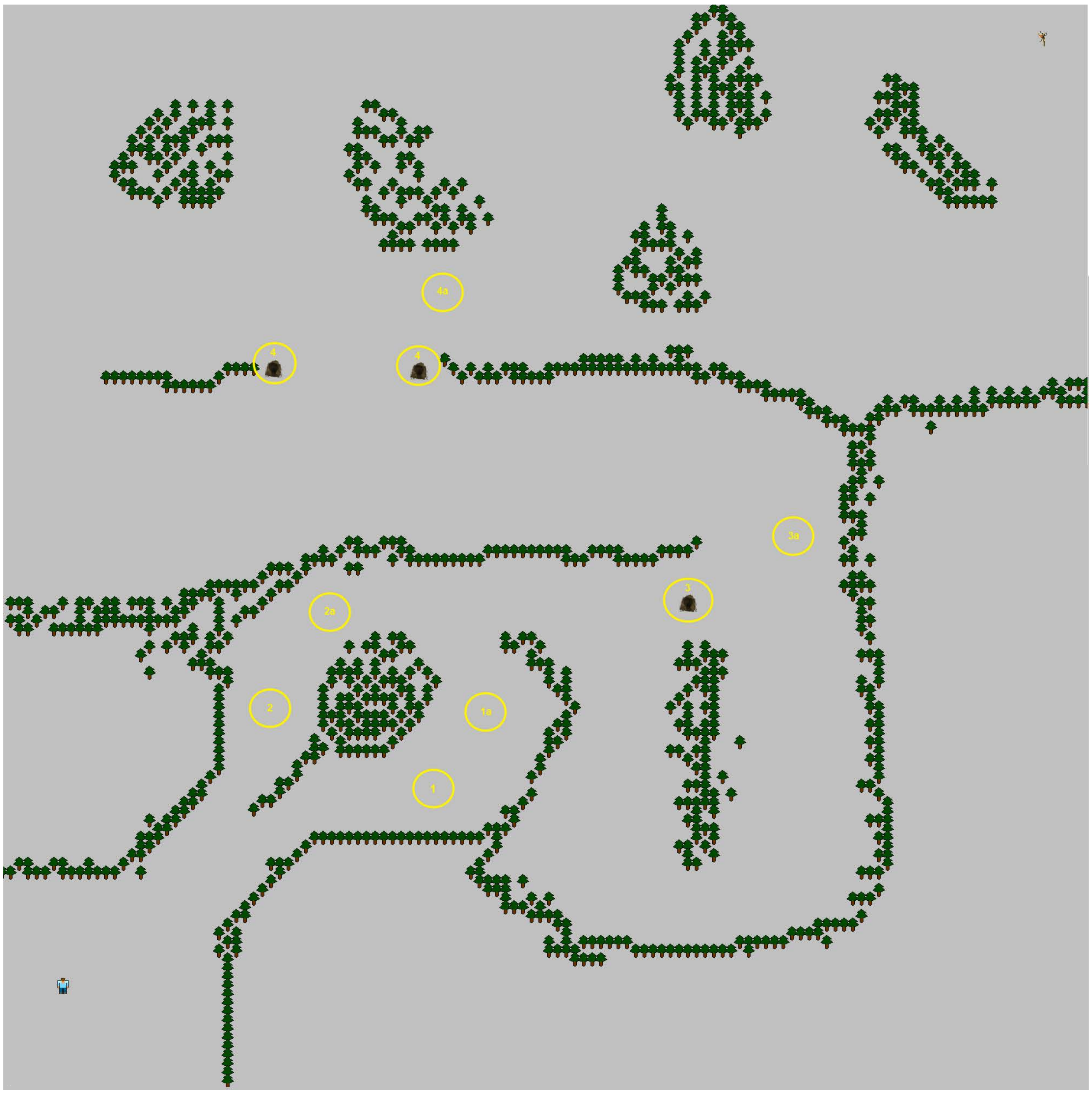


Figure 2

G | Map



H | Trainee Script

Trainee script

Trainee T1 – Inexperienced

[ENSURE THE APPROPRIATE TRAINEE TYPE (CRTL) AND THE OTHER TEAM IS SELECTED (UP ARROW)]

1. Go up at first junction.
2. When lion comes.
    a. Run.
    b. Let a few hits connect.
    c. Run into corner.
    d. Cannot move.
    e. Lion runs back. **[cautious of lions - the trainee now recognizes lions as a potential danger and will run away when he notices them]**
3. Go the other way.
4. Encounter lion pit
5. Approach lion pit.
6. When lions come. **[cautious of lion pits - the trainee now recognizes lion pits as a potential danger and will approach them more cautiously]**
    a. Go back (left).
    b. After reaching junction.
        i. Stand still. **[recognize “standing still” – the trainee now recognizes that lions do not attack if he is standing still]**
7. Go back.
8. Go down and avoid the lion pit.
9. Approach double lion pit.
10. Try to go past them (not precisely in between, but to the right).
11. Lions come from right pit. **[recognize lion pits and understands danger – trainee is now fully aware of lion pits and that lions will come out when he comes too close, he will try to avoid lion pits]**
    a. Run back a little and stand still.
12. Go around this time.
13. Finish.

Trainee 2 – Experienced

[ENSURE THE APPROPRIATE TRAINEE TYPE (CRTL) AND THE OTHER TEAM IS SELECTED (UP ARROW)]

1. Go up at first junction.
    a. Ignore first lion.
        i. If additional lions come, go back and around.
2. Go past first lion pit at the top.
    a. Ignore first 2 lions.
        i. If additional lions come, go down.

3. Go exactly in between double lion pits to avoid agro.
    a. Go directly right if no additional lion pit was spawned.
    b. Go up + right if an additional lion pit was spawned.
4. Finish.

I Questionnaire

ISM Questionnaire

Name: ☐ **Date:** ☐

		strongly agree	agree	neutral	disagree	strongly disagree	not applicable
1.	I had a good overview of the scenario (knowledge of the events) before I started with the first trainee.	☐	☐	☐	☐	☐	☐
2.	I had a good overview of the scenario (knowledge of the events) before I started with the second trainee.	☐	☐	☐	☐	☐	☐
3.	I had a plan formulated (an idea of how to respond to the possible behaviours of the trainee) before starting with the first trainee.	☐	☐	☐	☐	☐	☐
4.	I had a plan formulated (an idea of how to respond to the possible behaviours of the trainee) before starting with the second trainee.	☐	☐	☐	☐	☐	☐
5.	I was able to correctly and timely trigger events in the scenario during the game.	☐	☐	☐	☐	☐	☐
6.	I was able to correctly and timely add events to the scenario during the game.	☐	☐	☐	☐	☐	☐
7.	I had a good overview of all the events during the game.	☐	☐	☐	☐	☐	☐
8.	It was easy to trigger events during the game.	☐	☐	☐	☐	☐	☐
9.	I knew whether the trainee's response to events was correct or faulty.	☐	☐	☐	☐	☐	☐
10.	I knew how to respond to faulty behaviour of the trainee.	☐	☐	☐	☐	☐	☐
11.	I successfully used the scenario to help the first trainee reach the learning goals.	☐	☐	☐	☐	☐	☐
12.	I successfully used the scenario to help the second trainee reach the learning goals.	☐	☐	☐	☐	☐	☐
13.	I enjoyed using the game to educate the trainee.	☐	☐	☐	☐	☐	☐
14.	The game world (lions and lion pits) behaved according to my expectations.	☐	☐	☐	☐	☐	☐

J | Experiment results

Var	Median-U	IQR-U	Median-S	IQR-S	Sig
SCEN1_EVNT1	0	1	0	1	0.778
SCEN1_EVNT1A	0	0	0	0	1.000
SCEN1_EVNT2	1	0	1	1	0.611
SCEN1_EVNT2A	0	1	0	0	0.121
SCEN1_EVNT3A	0.5	1	0	0	0.088
SCEN1_EVNT4A	0	0	0	1	0.611
SCEN1_EVTCREATED	2	3	1	2	0.054
SCEN1_EVTADAPTED	0	0	0	1	0.081
SCEN1_SELFCREATED	0	1	1	2	0.245
SCEN1_SELFADAPTED	0	0	0	0	0.346
SCEN1_HPLEFT	9.5	11	7	3	0.146
SCEN1_TIMELEFT	139	47	109	7	**0.014**
SCEN2_EVNT1	0	0	0	0	0.931
SCEN2_EVNT1A	0	0	0	0	0.289
SCEN2_EVNT2	1	0	1	0	0.931
SCEN2_EVNT2A	0.5	1	0	1	0.824
SCEN2_EVNT3A	1	1	1	1	0.417
SCEN2_EVNT4A	1	1	1	1	0.896
SCEN2_EVTCREATED	3.5	4	1	7	0.109
SCEN2_EVTADAPTED	0	0	2	3	**0.002**
SCEN2_SELFCREATED	0.5	2	1	5	0.473
SCEN2_SELFADAPTED	0	0	0	1	0.082
SCEN2_HPLEFT	11	6	5	8	**0.020**
SCEN2_TIMELEFT	171	42	153	16	0.101

QUEST_OVERVIEWSCEN1	3.5	1	4	2	0.118
QUEST_PLANSCEN1	3	2	4	2	0.253
QUEST_PLANSCEN2	4	1	5	1	0.526
QUEST_CORRECTIMELYTRIGGER	3	2	4	2	0.487
QUEST_CORRECTTIMELYTADD	3.5	2	4	2	0.483
QUEST_OVRVIEWGAME	4	2	4	2	0.336
QUEST_EASYTRIGGER	3	2	4	2	0.100
QUEST_KNEWBEHAVIOUR	4	2	4	1	0.200
QUEST_KNEWRESPONSE	4.5	1	4	2	0.213
QUEST_TRAINEE1SUCCESS	3.5	2	4	1	**0.028**
QUEST_TRAINEE2SUCCESS	3.5	3	4	2	0.143
QUEST_ENJOYEDGAME	4	1	4	2	1.000
QUEST_EXPECTEDGAMEBEHAVIOUR	4.5	2	4	3	0.719