

On-line Adapting Games using Agent Organizations

Joost Westra, Hado van Hasselt, Virginia Dignum, Frank Dignum

Abstract—Almost all computer games that are currently created use fixed scenarios or simple fixed rules to define the course of the game, which mostly results in very predictable and inflexible behavior of all the elements in the game. Current research done on dynamic adjustability in games already makes it possible for different elements to adjust to the player. However, these approaches are still using centralized control. The serious games we are investigating are constructed using complex and independent subtasks that influence each other. Using centralized control becomes impractical if the complexity and the number of adaptable elements increase. We suggest a multi-agent approach for adapting serious games to the skill level of the trainee. Using separate agents makes it easier to guarantee the natural progression of each element of the game and thus its believability. The user task is selected based on a combination of the possible situations that can be provided by the agents at that stage of the game. The task selection is thus dependent on the user model, the agent preferences and the storyline of the game. The storyline and other requirements are specified by using an agent organization framework. An update function for the user model according to the performance of the trainee, the difficulty of the task and the amount of influence of each subtask is also given.

I. INTRODUCTION

Computer games are increasingly being used for training purposes. Games created with the purpose of training people are called *serious games*. These training programs need to be suitable for many different people and therefore the games need to be adjustable. Currently this adjustment is either done externally by experts that need to guide the adaptation, or the application has a number of predefined levels. On the one hand, the use of expert guidance to adaptation results in quite effective training. However such experts are rare and expensive, and it is not always feasible to have an expert available for each training session. On the other hand, the use of predefined adaptation levels may lead to less optimal adaptation in the case trainees do not fit well with the expected stereotypes. To optimize learning it is beneficial to (automatically) adjust the game while the trainee is playing without the need of an expert guiding this process. This adaptation during gameplay is called online adaptation. Even though many commercial games do not use any adaptation [1], already some research has been done on adaptation in games. However, most of this research focuses on adaptation of certain simple quantitative elements in the game. For example better aiming by opponents or adding more or a stronger type of opponents.

Joost Westra, Hado van Hasselt, Virginia Dignum and Frank Dignum are with Universiteit Utrecht, Padualaan 14, Utrecht, Netherlands (phone: +31-30-253-4432; fax: +31-30-253-4619; email: westra@cs.uu.nl, hado@cs.uu.nl, virginia@cs.uu.nl, dignum@cs.uu.nl).

There are three important aspects [2] that have to be considered when performing online adaptation. First, the initial level of the player must be identified. Second, the possible evolutions and regressions in the player's performance must be tracked as closely and as fast as possible. Third, the behavior of the game must remain believable. In this paper we will mainly concentrate on the third aspect while trying to achieve the first two. Believability has a lot different aspects, one of the main aspects is that the behavior of the characters should be long-term coherent [3].

Serious games usually follow stricter scenarios than commercial games, because they have to guarantee some useful learning experiences for the trainee. The ordering of different tasks for the trainee are thus more important than in commercial games. On the other hand, we want to provide the trainee with significantly different, possibly predesigned, believable tasks that are created to optimize learning (the second important aspect of online adaptation). However, allowing unlimited and unorganized adaptations quickly leads to a disturbed storyline and the believability of the game will be diminished. The characters and other adaptable elements have to remain consistent because they are usually active for relatively long periods in serious games.

For example, a serious game can provide a training task for a fire commander that needs to make sure the victims in a burning building are saved. This task is influenced by a lot of different characters. The victims could be more or less mobile or they could be located in simple or difficult locations. There may be bystanders that could obstruct the medics. The police could autonomously control the bystanders or could only act if ordered by the fire commander. Also the behavior of the medical personnel affects the difficulty of the task. As becomes clear from this example the user task is very dependent on the behavior of all the characters in the scenario.

Because the tasks that the user needs to perform follow from the behavior of all the different characters the behavior of the characters needs to adapt to the user skills. This means that the characters behave differently on the same task dependent on the skill level of the user. This gives a lot of different possible variations because for each subtask there are multiple variations and they can be combined in multiple ways. One difficulty in this adaptation is that the performance of these type of subtasks can not be measured separately because all the behaviors influence each other.

If all agents are allowed to adapt to the user without any coordination, situations will occur where all agents adapt at the same time and thus create unwanted scenarios for the trainee. For example, the victims become less mobile, the police agent becomes less autonomous and the bystanders

obstruct the personnel more, all at the same time. We do want to have emergent behaviors that adapt to the user but also want the emergence to stay within the boundaries specified by the designers. We want to be able to define a storyline in such a way that there still is room for adaptation while making sure that the storyline of the game is preserved. Finding a good trade-off between flexibility and control is a very common problem in the agent technology community. From this research we take the idea of using agent organizations as a means to mediate between individual autonomy and overall control.

In the next section, we will discuss the properties of serious games in more detail. Subsequently we define some requirements on the adaptation algorithms for on-line adaptation. In section IV we will discuss some related work on adaptation in games and properties of different learning algorithms and agents. In section V we propose our learning algorithm that uses pre-designed subtasks in combination with a dynamic user model. We also show how combined tasks can be selected that are both suitable for the user and are within the limits defined by the agent organization. We illustrate the use of the model with an example in section VI. Extensions and conclusions are discussed in the last two sections.

II. CHARACTERISTICS OF SERIOUS GAMES

As already discussed in the introduction, serious games have some different requirements than games that are only played for fun. One main difference is that a training with a serious game should not take more than a certain amount of time. For example a certain part of a course should not take more than a day. While for entertainment the game should just be enjoyable for as long as possible. Another difference is that serious games are usually better structured and thus easier to split up into different subtasks. This is because these games are created for testing different abilities of the user. Usually these subtasks are tested in a relative short amount of time and possibly repeated multiple times to see if the player has improved enough. A typical scenario is that in the beginning most tasks are done one by one to avoid overloading the user and when the training progresses, more tasks are done in parallel. This creates an extra load on the user and significantly complicates the task. This is an element that also has to be factored in when selecting tasks. This means that if the trainee has learned the different subtasks without having to do multiple tasks at the same time the difficulty of these subtasks increases.

The storyline of most serious games is usually also more complex than most commercial games. The trainee is supposed to achieve a certain skill level for a number of tasks before he is exposed to tasks that require these previously learned skills. We want the training to be as flexible as possible, meaning that we want to allow different orderings and combinations of the subtasks as long as it results in a legal ordering specified by the designer.

In most commercial games the characters do not change a lot over the course of the game. Often this is the case because

the player only interacts for a relatively short amount of time with these game characters. For example, the characters are shot by the player within a minute of interaction. They also usually do not change because the game is not adapting to the users. If characters do change their behavior this is done in a very prescribed fashion. In serious games it often happens that the player interacts with the game characters for relatively long periods of time. For example interacting with a team mate in a joint operation in which the trainee is the team leader. Characters are active for more than one measurable episode, these episodes should follow each other as smooth as possible. This means that from the perspective of the trainee, the whole training is one continuous task. Because we want the game to adapt to the performance of the user the characters are changing their behavior over time. The danger when adapting elements of the game that are observed by the player is that the player will notice if these elements adapt in an unnatural and unbelievable way.

The goal of a serious game is to teach certain prespecified tasks to the user of the game (the trainee). Because all users are different not all users learn optimally when providing them with the same challenges. We want to provide different challenges to different users. Besides having different users and different tasks there also is a storyline of the game. The storyline is the intended progress of the game as intended by the game designers. This means that we want to be able to model different users, select appropriate tasks for them and to guarantee the desired gameflow.

Different adaptable elements of the game will be responsible for teaching different tasks to the user. This means that if the user is able to adapt to certain tasks quicker than other tasks these elements have to adapt faster. Keep in mind that the behavior of all the adaptable elements in combination with the environment is what defines the task for the trainee.

III. REQUIREMENTS ON ADAPTATION

Because we want the agents to autonomously make decisions about what is believable and what fits the planned storyline of the game, a representation needs to be available to the agents.

Besides general ordering over time the different adaptable elements also influence each other per episode. A definition for the allowed interactions is therefore required. One could think of several situations where the selected actions would make sense for all the separate agents but that the combination gives undesired results. For example if all agents decide to perform a certain action in the same small room at the same time. Or the designer could have the intention that there is always one agent operating the fire engine pump but all agents decide that staying at the pump is not their best behavior. Agent organization frameworks are designed to regulate these kinds of interacting behaviors.

The difficulty adjustment also can be coordinated between the different agents using the organizational control. If the user is performing above expectation and all the agents decide to increase the difficulty at the same time the application will probably be too difficult the next time. So, the

organization will determine how many agents are allowed to adapt.

In training applications it usually is possible to make an estimation of the difficulty of a certain task. This can be done by using domain expert knowledge. When creating different possible task implementations for training, they are intentionally created with different known difficulties. These estimates can be updated in an offline learning phase and will get more accurate if more users have used the application. Having this information in advance significantly speeds up the adaptation because the algorithm not only needs to learn the strength relations between the different task implementations but is even able to directly select appropriate implementations for a given user skill level.

IV. RELATED WORK

In this paper, we advocate to bring together three aspects to adapt serious games to the user. The adaptation should be distributed over the separate elements that constitute the story line, these elements should adapt themselves online using some machine learning technique and they should do it in an organized fashion to maintain the general story line. In this section, we discuss the related work that can be used for these aspects.

A. Adaptation in games

Most commercial computer games that have varying degrees of difficulty do not use online adaptation. They have a number of preset difficulty levels that need to be selected in the beginning of the game. Offline learning could be used for these predefined levels but usually these are just scripted by hand.

Current research on online adaptation in games is based on a centralized approach [4], [5]. Centralized approaches define the difficulty of all the subtasks from the top down. This is only feasible if the number of adaptable elements is small enough and if the separate adaptable elements have no separate time lines that need to be taken into account. In shooting games, for example, these requirements are not problematic. The games only adapt to the shooting skill of the user and most characters only exist for a very limit period of time. However, in serious games there are usually a number of separate tasks that need to be learned. Also the characters and other adaptable elements in the game are active for longer periods of time and should adapt during their lifetime and adjust in a consistent and believable fashion.

Another important aspect is the distinction between direct and indirect adaptation. Direct adaptation is adaptation where the designer specifies possible behavior and how to change this behavior. The designer also specifies what input information should be used. When using indirect adaptation the algorithm optimizes performance by using feedback from the game world. This requires a fitness function and usually takes many trials to optimize. If indirect optimization is used the algorithm also needs to be able to cope with the inherent randomness of most computer games. In this paper, we will use an approach that has the benefits of direct adaptation

without the need for the designer to directly specify how the adaptation should be done. The designer is able to specify certain conditions on the adaptation to guarantee the game flow but does not have to specify which implementations are chosen after each state.

Some research has been done on using reinforcement learning in combination with adaptation to the user [4], [6]. Most of these algorithms rely on learning relatively simple (sub-)tasks. Moreover, the aim of these adaptation approaches is learning the optimal policy (i.e. making it as difficult as possible for the user). In order to avoid that the system becomes too good for the user, some approaches filter out the best actions to adjust the level of difficulty to the user. This results in unrealistic behavior where characters that are too successful suddenly start behaving worse again. Little attention is paid to preserving the story line in present online adaptation mechanisms, because they only adjust simple (sub-)tasks that do not influence the storyline of the game. Typical adjustments are for example changing the aiming accuracy of the opponents or adding more enemies. However, even when adding more enemies the algorithm should already take into account that they can only be added in unexplored areas and do not influence the progress of the game.

B. Machine learning

Not all learning algorithms are suitable for adaptation in computer games. An important factor is that adapting to the user in games can only be done while the user is playing the game [7], [8]. This approach is called online adaptation. This contrasts with offline learning where learning is done in an earlier phase based on previous prerecorded data, without direct input from the user. Online adaptation requires that the algorithm learns a lot quicker with a lot less episodes. Because the game is adapting while the user is participating in the game, it is important that no unwanted situations caused by the adaptation occur. This means that the adaptation should only try promising and believable solutions while exploring different options. In offline learning this is not important as long as the end result is good. Something else to keep in mind is that with online adaptation you do not want the algorithm to converge because the skill level of the user keeps changing. This means that for example the learning rate should never be decreased to zero.

In this paper, we will focus on not supervised learning (we use this term to avoid confusion with unsupervised learning). That is, we do not expect the user or an expert to specify the performance of the user on certain tasks. The algorithm also needs to be scalable, meaning that it also functions fast enough if a lot of adaptable elements are added. One of the most important factors when performing online adaptation is to limit the size of the state space. In offline learning the algorithms are usually given a lot of different input variables and the algorithm is given enough time and is flexible enough to learn which have the most influence. In online learning this usually takes too much time. The number of possible actions should also be small enough. It takes a lot of time to

find which actions are promising in which state. This means that either expert knowledge or offline learning algorithms should be used to identify which states are important and which actions are most promising. The algorithm should also be able to cope with incomplete information because not all information is available to the character to make sure it behaves believable. With these requirements in mind, certain algorithms can already be excluded. For example, evolutionary algorithms are not suitable for this task because testing different possible implementations would already consume a lot of time and this process has to be repeated for every generation. There are some extensions [9] that make it possible to do online adaptation but this process is still very slow if only one possible implementation can be tested at the same time.

Reinforcement techniques are more suitable but also can not be used without some limitations. We will base our approach on reinforcement learning but adapt it to our specific needs of reducing state space and high speed of adapting.

C. Agent organizations

Current most use of agents for gaming assumes centralized control over agents and furthermore agents have no learning capabilities which makes adaptation impossible. In systems that do allow learning are still restricted in the learning possibilities and also assume centralized control. Adapting the game to the user for complex learning applications requires both learning capabilities and decentralized control. However, in order to guarantee successful flow of the game and the fulfillment of the learning objectives, the system needs to be able to describe global objectives and rules. Although many applications with learning agents exist, multi-agent systems with learning agents are usually very unpredictable. This is not a problem for applications like simulations or optimizations where only the end result matters. But for (serious) games, where the agents are adapting during the game and thus the adaptation directly has to have a beneficial influence, the system needs to be a lot more predictable. Systems with this type of characteristics have been successfully modeled using agent organization frameworks such as OperA [10]. In this framework it is possible to define conditions when certain actions are allowed or not. The ordering of the different possible actions can also be defined in this framework. This allows the designer to make sure that the users are not exposed to tasks that are not suitable yet or would ruin the storyline. In previous work we have shown how to use agent organizations to specify the boundaries of the game [11].

The OperA model for agent organizations enables the specification of organizational requirements and objectives, and at the same time allows participants to have the freedom to act according to their own capabilities and demands. In OperA, the designer is able to specify the flow of the game by using landmarks. Figure 1 shows a very simple example of an OperA interaction structure. The different elements of the game are represented by scenes (the squares in the figures

are separate scenes) which are partially ordered without the need to explicitly fix the duration and real time ordering of all activities. That is, OperA enables different scenes of the game to progress in parallel.



Fig. 1. Interaction Structure

Such an interaction structure defines the ordering of the scenes and when it is allowed to transition to the next scene. The scenes are defined by scene scripts that specify which roles participate and how they interact with each other. The definition of the organization can be so strict that it almost completely defines the strategy. But it is also possible to specify the organization in such a way that all the agents in the game work towards achieving the goals of the game but are still able to do this using different strategies. In the scenes, the results of the interaction are specified and how and in what order the different agents should interact. It is also possible to define norms in the scene description. This makes it possible to put extra restrictions on the behavior of the agent. In a scene script, it is also possible to define certain time constraints to make sure that the game progresses fast enough.

D. Adaptation with BDI-agents

The characters of the game have it as their own goal to ensure that they follow a believable storyline. They also have to make complex decisions during to game because not all information is known before the game started. Using BDI agents is a good fit because it allows us to create intelligent characters that are goal directed and able to deliberate on their actions. For the implementation of the BDI agents we will use the 2apl [12] language. A disadvantage of using BDI agents is that they are usually not suitable for learning. However, in 2apl there is an easy approach in which there are multiple equivalent 2apl plans for each goal that are suitable for different skill levels. We have already created a 2apl extension [13] to make it possible to select different equivalent plans according to their corresponding weight. The weights are adapted to the user according to the feedback from the game.

V. OLA IJS

As we have shown in the previous sections the requirements we currently have are not resolved by current work on adaptation in games. In this section we propose an adaptation approach that is able to handle these requirements. The proposed system only uses not supervised learning because we want the system to operate autonomously. It will use expert knowledge for the adaptation but it will be more flexible than current direct adaptation methods. The story line will be defined by an agent organization framework.

Important to keep in mind is that the task that needs to be executed by the user follows from the behavior of the agents. Each agent is responsible for adapting its own subtask while making sure that believability is preserved. Think of the example where the trainee is a fire commander that needs to secure victims inside a burning building. An example subtask could be controlling the fire, this can easily be increased in difficulty by letting the fire grow more rapidly. However it would be unrealistic if the fire suddenly doubles in size without a gradual increase. There are a lot of other subtasks that also influence the performance of the trainee. The bystanders could be obstructing the medical personal more if time progresses but it would be unbelievable if the number of bystander suddenly doubles. A typical example of subtasks directly interacting with each other occurs when there is a police agent that has the ability to influence the bystanders. These are however subtasks from the training perspective. The trainee needs to give orders to the police man if he is not performing well while dealing with the bystanders is a different task.

A. task selection

As explained in the previous section the goal is to select the best tasks suiting the needs of the user. To make the adaptation as fast as possible we are using a flexible form of direct adaptation in the online phase. This means that all the different possible behavior variations are already implemented and stay fixed. From all these possible behaviors and combinations of behaviors of all the different agents we want to select the most suitable task for the user. Something to keep in mind is that the tasks that need to be performed by the user are dependent on the game environment but even more on the (adaptive) agents. These agents perform different plans given a different task preference. For example a task could be made easier if a cooperative agent is autonomously performing a large portion of the task without requiring lots of input from the user. The system is a dynamic adapting system because it is making estimates about the user performance during gameplay and using this information to select the most appropriate task. In subsection V-C we will discuss how the user information is updated.

The user model is not a very accurate representation of the user when the training just started. We will use a default user model which represents an average user that uses the application for the first time. From this user model we will select a task that is most suitable for the user but also preserves the gameflow. Part of the preservation of the gameflow is handled by the agents themselves.

Each agent has its own preferences and these preferences are likely to conflict with each other. We have chosen to use a kind of combinatorial auction [14] to select the best suitable plans for all the agents. The main objective that needs to be optimized by the auction is to choose the most optimal task for the user. The agents representing the different adaptable elements will make different possible variants of their subtasks available that can be executed in the next time step. Thus the task of the user is created by

combining the subtasks of the agents. The variants that are made available depend on the restrictions that are defined by the agent organization. The agents will be able to give preference relations between the different variants. The actual selection is thus dependent on the required task for the user and on the preferences of the agents. It is beyond to scope of this paper to show exactly how this is done.

Figure 2 shows a schematic overview of the whole task selection process. This is optimized on the skills of the user but also most fitting with the preferences of the agents. The preferences of the agents are different because they have as a goal to stay as consistent and believable as possible and to follow their own preferred ordering of tasks. The agent uses information from the OperA model where the ordering of tasks can be specified. During this task selection we also use the restrictions from the OperA model to select a fitting task according to the organizational requirements. An example of this is that in the organization it is specified that one agent always goes left and one agent goes right. In the case that one agent specified no preference while the other agent prefers to go right, the agent without a preference should select the variations where it goes left.

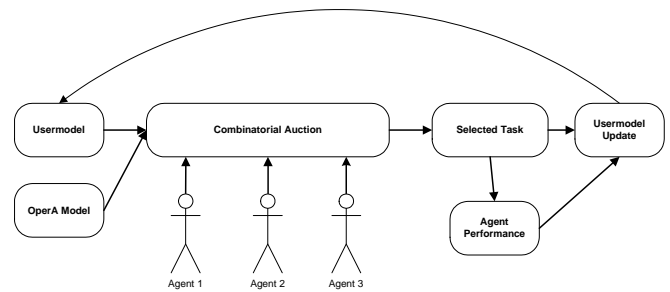


Fig. 2. Task selection

The resulting "selected task" is a combination of all the plans that will be executed by the agents. This task has a certain combined difficulty for all the separate learning tasks. After the task is completed this information can be used to update the user model. The update of the user model is also dependent on the performance of the user.

B. Task model

In this subsection we will formalize some important aspects of a training simulation. Of each of these aspects, we will discuss how to obtain the values of these. Most importantly, we will show how to automatically determine user models using task models and vice versa.

An important aspect is the task model. As a task model, we will use a tuple of length P , where P is the number of relevant properties a task can potentially have in a certain domain. A property typically will correspond to a certain task or subtask such as *extinguishing a fire* or *giving team orders*. The tuple of a task $T = (t_1, t_2, \dots, t_P)$ contains values between 0 and 1. These values indicate the expected needed skill of a user to perfectly complete the task. For instance, on a task with a value of 0.5 on some property, a user with a

skill level of 0.2 is expected to have difficulty completing the task, while a user with a skill of 0.6 should be able to perform the task perfectly. The assumption is that tasks with a skill level a little higher than the skill level of a user for certain properties can be expected to be challenging enough to be valuable for training, but not too hard to complete. Below we will discuss how such task models can be updated using training information, but for now we can consider these as given.

A second aspect is the task weights model. This is a tuple of length P : $W = (w_1, w_2, \dots, w_P)$ that indicates how much each skill is needed for the task. These weights will be used to determine how much to update each skill level of a user that performed the corresponding task. We require each element w_i of a weights tuple to be a real valued number $0 \leq w_i \leq 1$.

There are two equivalent ways to establish such a weights model in a useful way. In the first case, each weight corresponds to the amount of impact that each property has on the training as a whole. For instance, one way to determine such weights is to use the expected amount of time needed for a certain subtask corresponding to a certain skill. Note that this is an orthogonal concept to difficulty, because subtasks that make up the largest part of a task are not necessarily also the hardest parts of that task.

In the second case, a perhaps easier way to handle these weights tuples is to separate the impact on learning as a whole from the make up of a certain task. Then, the weights of a task could be required to sum to one over all skills. The interpretation of a weight then is the part of that skill that is needed to perform this task. Such tuple may be somewhat easier to construct, for instance by human experts. However, then still a indication of the size of the task compared to the training as a whole is needed, that should be stored in a different number V . Multiplying each element of a weights tuple in the second case with this value should result in a weights tuple such as described in the first case. In the remainder of this paper, we will therefore assume that the weights can be interpreted as the impact of that part of the task on the training as a whole, as described in the first case.

The third aspect is the reinforcement function. Such a function should be defined for each task, such that it can give feedback about the level of performance of the user on that task. The output of a reinforcement function is a number r , that can range from 0 to 1, where the interpretation is that when a user receives a reinforcement of $r = 1$ the task was performed perfectly. Conversely, a reinforcement of $r = 0$ indicates that the user failed to complete the task at all. We only consider reinforcement functions that are constructed by human experts, although for most domains some metrics such as the time needed to complete a task compared to the average time needed to complete the task by similar users could be used as reinforcement.

Finally, as the fourth aspect, we consider the user model. This model is also a tuple of size P : $U = (u_1, u_2, \dots, u_P)$, where each element indicates the expected skill level of the

user on each corresponding skill. This model is hard to obtain a priori, since it would require a screening of each user on all the relevant skills. Therefore, in the next section we will discuss ways to update this model automatically.

C. Updating User Models

As mentioned in the previous section, a user model is a tuple of size P containing estimates of skill levels for the user. For instance, these could be instantiated with a value of 0 for each skill, meaning we do not expect the user to be able to perform any skill to a desired level yet.

We now allow a user to try to perform some scenario. This scenario will consist of a number of tasks that can potentially be divided further into subtasks. For each task we assume a task model, a weights model and a reinforcement function exist. After a user completes any subtask for which each of these aspects is defined, we can update the user model as follows:

$$\forall u_i \in U : \quad u_i = (1 - \alpha_i w_i) u_i + \alpha_i w_i r \max(t_i, u_i) \quad , \quad (1)$$

where $0 \leq \alpha_i \leq 1$ is a step size learning parameter. First assume that $u_i < t_i$. Since we required that w_i is between 0 and 1, the effect of this update is as follows: if the task was performed perfectly, $r = 1$ and u_i will get updated towards t_i with a step size dependent on α_i and w_i . If, on the other hand, the task failed completely, the update results in an update of u_i towards zero. It can be easily verified that for skills that are not needed for the task the user skill is not updated, since w_i is then equal to zero. Typically, we will want α_i to be reasonably high in order to learn quickly from the received reinforcements. It is however possible to set α_i to zero, for instance when the main goal of a certain task is to only update specific parts of the user model.

When $u_i > t_i$, the user is assumed to be able to perform a task with difficulty t_i perfectly. If that is indeed the case, $r = 1$ and the user is updated towards u_i . This implies the user model is not updated. However, if the user is not perfect and $r < 1$, the user will get a negative update that is dependent on how low the reinforcement in fact is. This update should ensure that the skill levels in the user model will converge to the actual skill of the user.

We note that the expected user skill u_i will get updated positively every time that $r t_i > u_i$. This implies that for difficult tasks, a user with low skill does not need to perform perfectly to increase the expected skill. In general, we want users to increase their skill levels, so we want them to perform tasks where the expected reward is higher than u_i/t_i . If we are correct in estimating these expected reinforcements, we can easily determine which tasks are fruitful options for training. This brings us to the following important point: estimating the expected reinforcements.

D. Estimating Expected Reinforcements

As stated in the former section, if we have an accurate estimate of the reinforcements for a certain user on a certain task we can more easily determine which tasks have a good probability of increasing the users skill and are therefore

TABLE I
ORGANIZATION SPECIFICATION

Fire Agent Goal:	Simulate fire
Cooperative Agent Goal:	Listen to commands
Norms:	Stay believable
Landmarks:	Extinguish ground fire , Extinguish ceiling fire

suitable choices for training. In general, we can already determine that tasks with a lower difficulty than the skill of the user are never suitable, since then for all possible values of r , we have $rt_i < u_i$. However, some tasks will be too difficult, resulting in a high probability that $r = 0$ and therefore $rt_i < u_i$ and again the user skill will be updated negatively. Therefore, we are looking for tasks where $t_i R(U, T) > u_i$, where $R : U \times T \rightarrow [0, 1]$ is a function that gives the expected reinforcement for a given user on a given task. For any user, it is then possible to determine the expected reinforcement and therefore the expected update to the user model.

There are many ways one can estimate a function. We note that we assume that enough information on the expected reinforcement is contained in the user and task models. This means that the actual function may be somewhat noisy, but since the user model itself is also noisy, and we only intend to use it as an indication this should not be a problem. One way the function may be approximated is by use of supervised learning techniques such as a neural network. This network could then take all user model and task models that have already yielded a reinforcement as training inputs and the actual reinforcements as output. This general function can then be updated for all users whenever more information is received and can therefore be expected to give reliable generalizations reasonably quickly.

VI. EXAMPLE

In this section we show with a simple example how the system works. The type of serious game we will look at is a simplified training for a fire commander. We will use two very different adaptable elements in this example. One important element in fire commander training is the behavior of the fire. The agent that controls a certain element for the adaptation does not always need to be a character in the game. Another important element is the behavior of the team member, in this example we will only use one team member. For simplicity we will only focus on two learning goals for the trainee: fire extinguishing and giving team orders. In table I we show possible agent organization specifications.

As we discussed in section V-C the training starts with a default user profile which is either designed by experts or by offline learning on data of multiple previous users. Lets assume the user has a proficiency of 0.5 for both tasks.

This user model information is then used by the task selection model in conjunction with the desired restrictions and preferences both on the agent level and on the organizational level. Because there are no prior actions performed

TABLE II
USER MODEL UPDATE INPUT DATA

User Model:	0.5	0.5
Task Model:	0.6	0.6
Weight Model:	0.4	0.6

by the agents it is easier to keep their behavior believable in this phase. For example the fire could start at any level of intensity before the user arrives and the cooperative member could be as intelligent as you like. There could however be some restrictions imposed on the organizational level. For example it could be specified that the trainee always first has to learn to extinguish a certain type of fire before he is exposed to any other types. The selection itself is done by a form of combinatorial auction as discussed in section V-A.

Lets assume that the first evaluation on performance is done when the trainee has finished extinguishing one room. When this period is over we get feedback about the performance of the trainee. These performance measures are always very domain specific and need to be designed by an expert. Experts creating these kinds of training are used to create performance measurable tasks. An example of a performance measure could for example be the time it took to extinguish the fire (in reality this would be more complicated). Remember that the we are getting the performance of the user for the complete tasks because different subtasks influence each other making it impossible to accurately measure independent performance.

We assume that the performance of the user was 0.9. After receiving this feedback the user model can be updated. We have the old user model $U=(0.5,0.5)$. After we had selected the tasks we also obtained the corresponding task model specifying the difficulty of the separate subtasks (lets assume this is $T=(0.6,0.6)$) and the appropriate weight model indicating how big the influence of each subtask is (lets assume this is $W=(0.4,0.6)$). In this example we will use a learning rate of 0.3 for all subtasks. Using this information (summarized in table II) we can update the user model using the formulas discussed in section V-C.

Task fire extinguishing:

$$(1-0.3*0.4)0.5+0.3*0.4*0.9*\max(0.5,0.6) = 0.5048 \quad (2)$$

Task team orders:

$$(1-0.3*0.6)0.5+0.3*0.6*0.9*\max(0.5,0.6) = 0.5072 \quad (3)$$

The resulting new user model is $U=(0.5048,0.5072)$ and is ready to be used for the next task selection. As you can see both the skill levels are updated positively because the user performed well on a task that is more difficult than the previously expected user skill level. Also note that the subtasks that had the highest influence in this task gets updated stronger. Using the new user model a new task needs be selected. Now the preferences of the agents are a lot stronger because it would be very unrealistic if the fire suddenly increases or decreases a lot in intensity. The

cooperating team member can also not suddenly become a lot less intelligent. Already in this very simple example it quickly becomes clear that it is a much better approach to have the separate agents be responsible for their own believability.

VII. FUTURE WORK

A. Updating Task Models

Up until this point we assume preconstructed task models. However, it is possible to also update the task models, based on experience with users. In particular, after a number of users with different skill levels for a subtask i have performed tasks that contain this subtask, the difficulty of this subtask can be evaluated.

Recall that the difficulty t_i of a subtask should be interpreted as the skill level with which a user can perfectly perform the task. One option would be to simply take the average skill of all users that performed the task perfectly. However, especially for simple tasks this would overestimate the difficulty, because there could be many users with skill levels far above the necessary minimum to perform the task perfectly that will all also get perfect scores. Another possibility would be to take the user with the lowest skill level to perfectly complete the task and take its skill level as the new task difficulty. However, this will usually underestimate the difficulty, since the user models have to be learned and a user could have a higher skill level than its user model indicates. To find a better way to estimate the task difficulties, we look at how the user models were updated after performing a certain task T . We assume that the user models as a whole are reliable estimates of skill.

$$t_i = \frac{1}{N} \sum_{n=0}^N r_i^n \max(t_i, u_i^n) . \quad (4)$$

This update takes into account all users, not just the ones that performed the task perfectly and will therefore not suffer from the overestimation as described above. Similarly, because it is not dependent on a single user that got a perfect reinforcement, but with an unreliable skill level, it is also not subject to the underestimation we described.

VIII. CONCLUSION

In this paper we have shown that in serious games, where a complex task is taught, the adaptation of the game to the skill level of the trainee is important. The goal is to have the application adapting to the user while he is playing the game. We have shown that using a centralized approach to adaptation in these applications becomes very impractical. It is much more natural if the different elements are implemented by separate software agents that are responsible for their own believability. The system does not only need to be flexible, the designer also must be able to define the storyline and put certain restrictions on the combined behavior of the agents. Agent organization frameworks are very suitable for creating a flexible system that still must follow a certain progression and behave according to certain norms. So, it seems the use

of agent organizations is a good choice for creating organized adaptation.

We have proposed a task selection system that uses the user model, the preferences of the agents and uses the guidelines from the organization. We also developed an algorithm that updates the user model according to the performance of the trainee, the difficulty of the task and the amount of influence of each subtask. We have shown by a very small example how the system works in practice. And we have proposed some future work that suggests using the information gathered by all the users to update the subtask difficulty specifications. Finally a framework for reasoning agents that are able to select different variations of subtasks, the learning 2apl agents, is also presented. The next steps will be the actual testing of a complete serious game on firefighting with actual firefighters in the Netherlands and a professional simulation environment provided by VSTEP. This is set up to take place in the near future.

ACKNOWLEDGMENTS

This research has been supported by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

REFERENCES

- [1] S. Rabin, *AI Game Programming Wisdom*. Charles River Media, 2002.
- [2] G. Andrade, G. Ramalho, A. S. Gomes, and V. Corruble, "Dynamic game balancing: An evaluation of user satisfaction," in *AIIDE*, J. E. Laird and J. Schaeffer, Eds. The AAAI Press, 2006, pp. 3–8.
- [3] D. Moffat, "Personality parameters and programs," in *Creating Personalities for Synthetic Actors, Towards Autonomous Personality Agents*. London, UK: Springer-Verlag, 1997, pp. 120–165.
- [4] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, "Adaptive game AI with dynamic scripting," 2006. [Online]. Available: <http://dx.doi.org/10.1007/s10994-006-6205-6>
- [5] R. Hunicke and V. Chapman, "AI for Dynamic Difficulty Adjustment in Games," *Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence (AAAI'04)*.
- [6] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Extending Reinforcement Learning to Provide Dynamic Game Balancing," *Reasoning, Representation, and Learning in Computer Games*, 2005.
- [7] C. Beal, J. Beck, D. Westbrook, M. Atkin, and P. Cohen, "Intelligent modeling of the user in interactive entertainment," *AAAI Spring Symposium on Artificial Intelligence and Interactive Entertainment*. Stanford, CA, 2002.
- [8] J. Chen, "Flow in games," *Communications of the ACM*, vol. 50, no. 4, pp. 31–34, 2007.
- [9] K. Stanley, B. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 6, pp. 653–668, 2005.
- [10] V. Dignum, "A Model for Organizational Interaction: based on Agents, founded in Logic," *SIKS Dissertation, series*.
- [11] J. Westra, F. Dignum, and V. Dignum, "Modeling agent adaptation in games," *Proceedings of OAMAS 2008*, 2008.
- [12] M. Dastani and Meyer, J.-J. Ch., "A practical agent programming language," *Proceedings of ProMAS 2007*, 2008.
- [13] E. Kok, "Learning Effective Rule Selection in 2APL Agents," 2007.
- [14] T. Sandholm, "Algorithm for optimal winner determination in combinatorial auctions," *Artificial Intelligence*, vol. 135, no. 1–2, pp. 1–54, 2002.