University of Utrecht

Master's Thesis

ICA-4198557

Feedback in resource management games based on problem-solving strategies

Author:
Krijn Thuis

Supervisor:
prof. dr. Johan Jeuring

January 11, 2016

Contents

1 Introduction

1.1 Serious games

Over the past decades serious games have become a new possibility to learn or train. What exactly is a serious game? Crookall [1] states that, even for the known concept 'game', there is no consensus about the definition. For the term serious game Crookall prefers *computerized simulation/game for training or learning.* The term serious game is widely criticized, since it would imply that these 'games' are serious instead of fun. Zyda [2] defines a serious game as 'a mental contest, played with a computer in accordance with specific rules, that uses entertainment to further government or corporate training, education, health, public policy, and strategic communication objectives'. In this thesis the widely used term serious game (or *SG*, in short) is used.

Since the field of serious games is rather new, an incomplete selection of developments in various fields will be summarized in the following section.

1.2 Types of serious games

Some of the most common serious games are games for *rehabilitation* and *exercise* games (sometimes called "exer-games"). For rehabilitation one could think of dexterity rehabilitation [3] or stroke rehabilitation [4]. For exercise games the main goal is to stimulate movement for a certain target audience, like elderly [5], or obese people [6, 7].

In the *medical* field serious games are often used to treat patients or to simulate difficult procedures for doctors or surgeons. SGs exist for disorders like Alzheimer [8, 9] or eating disorders [10]. For physician training many SGs exist, like knee replacements [11, 12], surgery simulations [13], and SGs about cancer genetics [14].

Because many researchers expect that SGs probably work well with *autistic* people, several games have been developed for this target audience. One of the characteristics of the autism diagnosis is the lack of emotion recognition and the expression of emotions. SGs can help to learn to copy emotions [15] or, train emotions [16].

As more and more attention arises for the *energy consumption*, many SGs have been developed in this area. SGs to save energy in households [17], to save energy in office environments [18], to stimulate energy awareness [19], and to inform people about smart grids [20], are examples of this development.

For business settings SGs can be used to regulate the emotions in finance decision-making [21], to improve negotiation skills [22], or to improve coaching skills [23]. Dangerous situations in working conditions, like a fire drill [24], can also be simulated.

Most of the above games simulate a problem in the real world and have a learning goal for the learner. The learner is a person who plays the serious game, and is considered the target audience. 'Good' games provide the learner feedback on his or her actions, so that the learner knows his or her progression and can improve his or her performance [25]. This feedback is meant to support the learner in the process.

1.3 Feedback

Several kinds of feedback play a role in serious games. Low-level sensory information, like tactile, auditory, or visual feedback, provides perceptual feedback in serious games. Think of a beeping sound when a key is pressed. The feedback (*output*) given by the serious game is the result of the *input* of a learner. We focus on feedback in serious games in an educational context.

Hattie et al. conceptualize feedback as *'information provided by an agent regarding aspects of one's performance or understanding'* [26]. An agent can be a teacher, book, peer, or a computer. Feedback thus is a consequence of performance, and is used to reduce discrepancies between current understandings and performance and a goal. To reduce discrepancies three feedback questions are identified: Where am I going? (Feed Up) How am I going? (Feed Back) and Where to next (Feed Forward)?

According to Jaehnig et al. [27] the content of the feedback can contain knowledge in several ways. The authors present several types of feedback, like: knowledge of results (e.g. *"correct"* or *"wrong"*), knowledge of correct response (e.g. *"Wrong, the answer is C"*), and elaboration feedback (e.g. *"Wrong, the answer is C, because X and Y increase over time."*). They argue that elaboration feedback is more effective than the knowledge of correct response, but may require more implementation time.

Dunwell et al. [28] decompose feedback in two elements; timing and content. The efficacy of learning transfer depends on these elements, and suggests a strong relationship between the two. The complexity of the content of the feedback is dependent on the frequency the feedback is given. The authors distinguish feedback types (in increasing complexity) like evaluative, interpretive, supportive, probing, and understanding. They argue that higher levels of feedback, like understanding feedback, require more technical ambition, and may require involvement of other actors (e.g. an instructor).

The literature does not agree on the effectiveness of the timing of feedback [29]. According to Shute et al. researchers disagree whether direct feedback, or delayed feedback is more effective on learning outcome and efficiency.

Taylor et al. [23] note that appropriate methods of assessment or proof of learning, are essential in game-based coaching. Two main types of assessment are summative and formative assessment. Summative feedback is used to evaluate learning in the form of a grade, while formative feedback can provide ongoing substantive feedback during training. These two types differ in purpose and can both be useful in game-based learning.

We observe that the content and the timing are two important properties of feedback in serious games. The content can differ in complexity, and there is no general consensus about the timing of feedback. Furthermore the content and the timing are interrelated, and thus must be considered jointly.

1.3.1 Intelligent Tutoring System

A notable system to provide feedback or instructions to a learner is an intelligent tutoring system (or ITS, in short). Nkambou et al. describe the goal of an ITS as

providing tutorial services that support learning [30]. One of these tutorial services is feedback. An ITS generally provides services to a learner without intervention of a human instructor. According to Nkambou et al. [30] some of the main aspects of an ITS are:

- An expert knowledge model, which contains the concepts, rules and problem-solving strategies;
- A student model, which models the learner's problem solving capabilities;
- A tutor model, which keeps track of the students progression, provides feedback and, decides what is and is not understood by the learner;
- And a user interface, needed to interact with the process.

The first three aspects are three different 'models': the expert model, the student model, and the tutor model.

The expert model, or domain model, *"represents skills and expertise that an expert in a particular domain holds"* [31].

The student model is the diagnosis of the process of the student. It contains the information about what a learner knows about the domain. In general it is important for learners to use a correct model of the problem [32], instead of using fudge factors, or other model inconsistencies. Fudge factors are factors to artificially solve a problem without a correct model of a problem. A learner could use a trial-and-error approach to construct a model that works in some conditions, but fails in other conditions. Because students may have inconsistencies about the model they should use to solve the problem, feedback can be of great importance.

The tutor model determines the understanding of the domain, and could generate feedback for the learner. Measuring the understanding of the domain by the learner can only be done by observing his actions or input, since we cannot monitor his actual mental state.

1.3.2 Feedback loop

In Figure 1 a feedback loop in a serious game is given. In this loop the learner tries to achieve a learning goal. The learner gives input, which is presented as an action to the serious game. This action affects the game state while considering the rules of the game and the learning goal. Using the rules of the game some abstract representation of feedback can be generated, which can then be translated into human-comprehensible feedback.

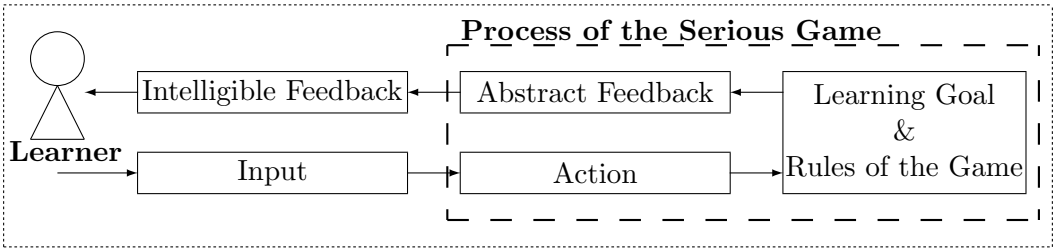

Figure 1: Example of a feedback loop

Imagine a game where the goal is to construct a city that runs on renewable energy. The learner selects an action which builds a coal plant. The serious game processes this action, and constructs abstract feedback. This action can result in abstract feedback that tells the learner that his or her action is not environmentally friendly and thus undesired. The learner now receives feedback in the form of video, audio, or text. The learner can use this feedback to choose a new action, and the loop continues.

1.4 Contribution

The main goal of this thesis is to describe an approach to provide feedback in serious games described with problem-solving strategies. "A problem-solving strategy is a technique that ... serves as a guide in the problem-solving process". [33] The type of serious games we will focus on in this thesis is resource management games. Think of games where the learner learns to build a city [34, 35], builds a city running on renewable energy [36], learns about the survival of animals [37], tries to maintain a government [38], tries to manage a soccer team [39, 40] or learns about economics.

Most resource management games have an underlying mathematical model. The mathematical model of the serious game should be completely known, meaning that the mathematical rules should be transparent and clearly defined at any moment in the simulation.

We propose an approach that uses the input of the learner to provide feedback based on problem-solving strategies. The approach takes two components as input: a problem and a strategy to solve this problem.

Based on the above description, the main research question of this thesis will be:

"How can we provide feedback based on problem-solving strategies using the input of a learner in resource management games?"

To answer the main research question, we need to know how we can provide feedback at any moment in time, and how we are able to provide a variety of levels of feedback and hints [41].

The rest of this thesis is organized as follows. In Section 2 the methodology will be described. In Section 3 the general approach will be described with some exemplar elaborations along with the requirements for the approach. Two example games will be discussed Section 4. Section 5 describes the results, and Section 6 concludes the thesis with our findings.

2 Method

This qualitative study explores the provision of feedback in resource management games. This study will describe the needed components to generate feedback in resource management games. We describe the necessary components of existing resource management games that are needed to generate feedback. The generation of feedback is based on a method by Heeren et al. [42, 43]

To show the applicability of the approach we study two cases in the form of conceptual games. The games are based on existing resource management games, but are an interpretation and a simplified version of such games. To demonstrate how feedback can be generated we describe use cases.

We structure our conceptual games similar to existing games and use the analogy between these two to answer the research question.

3 Components of our approach

In this section we describe an approach that provides feedback in resource management games. This feedback is based on problem-solving strategies. To ensure the applicability of the approach we define the necessary components and their restrictions.

3.1 The problem

To provide feedback to the learner, we first need to define the problem of a game. Resource management games involve the management of game resources, like natural resources, money or buildings. The learner has influence on various resources of the current game state. To finish the game the learner must achieve a goal state under certain conditions or rules.

In our approach the problem should be structured as an initial state, the actions that the learner can perform, an underlying mathematical model, and a goal. The model of the problem can be compared to the expert model in an ITS (see Section 1.3). The structure of a problem is defined, similar to the well-defined problem by Russel et al. [44], using the following components:

- **Initial state**: The initial state is a structured representation consisting of objects and relationships between these objects. Think of a game where the learner starts with resources and buildings.
- **Actions**: A description of the possible actions that can be executed in the form of a fixed set of actions $\{a_0, a_1, \cdots, a_n\}$. Think of actions that construct new buildings using resources.
- **Model**: The model which defines the rules which affect the initial state while taking the actions into account. Think of the depletion of resources over time, and not allowing the player to have negative resources.

- **Goal**: The goal that must be reached. It should be reachable with regard to the model and the actions. A goal state is a state that satisfies the goal constraints.

We structure a problem with the above four components to form a strategy. A strategy chooses actions to alter the game state from the initial state to a goal state.

3.2 Strategies

A strategy is a procedure that contains instructions to guide the problem-solving process. Since we want to provide feedback on the problem-solving attempt of a learner, we will define a problem-solving strategy. Heeren et al. [42] define a strategy as a *'procedure that describes how basic steps may be combined to solve a particular problem'*. Our problem consists of the four components in Section 3.1, meaning that is consists of an initial state, a set of actions, a model and a goal, as input.

The only input that a learner can give to solve the problem are actions. When the goal in a game is to generate energy, then the learner should, for example, build power plants to solve the problem. The strategy therefore must contain the possible actions, which are the smallest building blocks of the strategy. The main requirement of a strategy is to reach a goal state, using the set of actions in a particular order.

Serious games can consist of multiple strategies combined. Heeren et al. [43] describe a strategy language with a context-free grammar to specify step-wise procedures for exercises in domains like mathematics, logic or physics. Since we want strategies that define step-wise procedures, we choose their method to compose a strategy. Using the strategy language we are able to automatically calculate procedural hints and give an indication of the learners progression. Heeren et al. divide the main strategy in sub-strategies, which makes it possible to give feedback at any moment in the strategy. A (sub-)strategy can exist of *state constraints*. State constraints are criteria for a state. When, for example, energy is low (a state constraint), one should generate more energy.

Heeren et al. [42] define a strategy language using different combinators. Using this strategy language they are able to report advanced feedback about (sub-) strategies. Using a strategy language (discussed in Section 3.3) in combination with state constraints, we are able to construct feedback.

3.2.1 Sub-optimal strategies

A learner can choose actions because the action makes the game look nicer, or because it changes the game experience. In a real-time game a learner could also perform actions at a time that is not optimal. Even if the learner would take the optimal action sequence, the timing of the learner may not be optimal. To detect and provide feedback in such situations, sub-optimal or faulty strategies should also be described.

3.3 Strategy language

Now that a problem and a strategy have been examined, feedback can be constructed. Heeren et al. [42] point out it is important to integrate the feedback and the strategy, since they can not be seen apart. Heeren et al. define several combinators to describe a strategy using a strategy language. The strategy language of Heeren et al. contains the combinators in Table 1.

Combinator	Description
$s <*> t$	first s, then t
$s <\|> t$	either s or t
$s <\|\|> t$	s and t in parallel
$s \triangleright t$	apply s, or else t
while c s	apply s as long as c holds
label l s	attach label l to s
check c s	if condition c is true, perform s
$\in$	succeeding strategy
$\emptyset$	failing strategy

Table 1: Some strategy combinators by Heeren et al. [42, 43]

3.4 Feedback

To give feedback at the right time with the right content a strategy must be described. The strategy language by Heeren et al. makes it possible to label every part of the strategy. This makes it possible to give feedback at any point in the strategy. The resulting form of feedback will be a sequence of the (sub-)strategy labels. Think of a game where the goal is to obtain oil using pump-jacks, but to obtain oil the learner must first generate energy. This strategy should contain three sub-strategies, namely the generation of energy, the construction of pump-jacks, and obtaining oil. To perform the 'obtaining oil'-strategy, the constraints of the other two sub-strategies must first be met by executing the sub-strategies.

One advantage of labeling a strategy is that we can choose what labels we present to the learner. In the game described above the most shallow feedback would only contain the 'construction of pump-jacks'-label. If more feedback is needed, the 'generate energy'-label can be added. One could even label the specific action that the learner should perform. For a more in-depth example, see Section 4.1.

3.4.1 Detecting the learners strategy

To give feedback on the actions chosen by the learner, the strategy followed by the learner must be recognized. We can parse the learners action sequence against a described strategy, and we can indicate what actions of the learner are different from the actions that the strategy advises.

Let's give a simple example using a strategy containing three actions (a_0, a_1 and a_2) and two resources (r_0 and r_1). Consider the following three actions as given:

```
data Resources = R R1 R2

type R0 = Integer
type R1 = Integer


a0 :: Rule Resources
a0 = makeRule "a0" f
  where
    f :: Resources -> Maybe Resources
    f (R r0 r1) = Just (R (r0+2) r1)

a1 :: Rule Resources
a1 = makeRule "a1" f
  where
    f :: Resources -> Maybe Resources
    f (R r0 r1) = Just (R (r0-1) (r1-1))

a2 :: Rule Resources
a2 = makeRule "a2" f
  where
    f :: Resources -> Maybe Resources
    f (R r0 r1) = Just (R r0 (r1+1))
```

For every action we define the effect on the resources using a rule. Action a_0 increases resource r_0 with 2, action a_1 decreases both resources with 1, and action a_2 increases resource r_1 with 1. Now define the following strategy:

```
main :: Strategy Resources
main = check (\(R r0 r1) -> r0 < 2) <*>
          a0 ▷
             check (\(R r0 r1) -> r1 > 1) <*>
                a1 ▷
                a2
```

When the first resource r_0 is smaller then 2 the strategy takes action a_0. If resource r_0 is not smaller then 2, the strategy checks the second condition: '$r_1 > 1$'. If this condition is true the strategy takes action a_1, else action a_2. This strategy will always result in an action.

To provide feedback we now label the strategy. We can then use the labeled strategy to generate hints:

```
main :: LabeledStrategy Resources
main = label "Main strategy" $
       label "Condition 0" $      check (\(R r0 r1) -> r0 < 2) <*>
       label "Action 0"           a0 ▷
       label "Condition 1"           (check (\(R r0 r1) -> r1 > 1) <*>
       label "Action 1"              a1 ▷
       label "Action 2"              a2)
```

The above strategy contains labels to provide feedback. When for example the first condition is true, the strategy performs action a_0. When requesting a derivation

[45] with a game state using the above strategy, it would give the following label sequence:

```
["Condition 0", "Action 0"].
```

These labels can now be translated in the form of a hint. The learner can for example receive feedback like "The first condition is true." (`Condition 0`), and "You should perform the first action." (`Action 0`).

Besides generating a hint we can also provide feedback on the action chosen by the learner. Let's assume that the first check succeeds. The learner could take action a_2, while the strategy would take action a_0. The feedback should contain information about the the incorrect action. Every action is subject to checks on resources, and can thus not be performed when these checks fail. To catch these actions we define a 'buggy' rule for every action using the negation of the checks on the resources in the strategy. For the above strategy the three 'buggy' rules are:

```
a0' :: Rule Resources
a0'= buggyRule "buggy a0" f
  where
    f :: Resources -> Maybe Resources
    f (R r0 r1) = if r0 ≥ 2 then
       Just (R (r0+2) r1) else Nothing

a1' :: Rule Resources
a1'= buggyRule "buggy a1" f
  where
    f :: Resources -> Maybe Resources
    f (R r0 r1) = if (r0 ≥ 2 || r1 ≤ 1) then
       Just (R (r0−1) (r1−1)) else Nothing

a2' :: Rule Resources
a2'= buggyRule "buggy a2" f
  where
    f :: Resources -> Maybe Resources
    f (R r0 r1) = if (r0 ≥ 2 || r1 > 1) then
       Just (R r0 (r1+1)) else Nothing
```

When a learner chooses an action and the corresponding buggy rule succeeds, we provide feedback based on this buggy rule. For a game with a more practical example, read on to Section 4.1.

3.4.2 Goal-guarantee

Since the goal of the feedback is to advise a step towards the goal, we need a guarantee that the goal can be reached. So given a problem, there must exist a sequence of actions that results in a goal state.

In a game like tic-tac-toe providing feedback is easy, since we can create a tree of all possible states. Using this perfect strategy we can advise a next move to the

learner using the Minimax algorithm [46]. A perfect strategy in tic-tac-toe means that there is always an optimal move resulting in a draw, or win.

In some situations a perfect sequence of actions towards the goal state is impossible to find, or has impractical computation times. One could think of advising an optimal action in chess. In such cases a performance measure (for example a utility function [44, 47]) is a possible way of estimating the progression towards the goal.

For simple resource management games with resources that increase or decrease linear over time we could, for example, use linear programming [48] to calculate a goal guarantee (see Section 4.1.2 for an example). For more complex underlying mathematical models this can however be problematic. In this case we can use a strategy to provide a goal-guarantee. We can provide this goal-guarantee, if the strategy is able to derive a sequence of actions that reaches a goal state.

4 Example games

The previous section explains the general idea of giving feedback using a problem definition and strategies. In this section two imaginary games are presented to explain how this approach would work in practice. The first, simple game consists of a few actions, resources, and a strategy. Before the final strategy of this game is discussed, some sub-optimal strategies are presented, followed by improved strategies. The second game is an example of a game that has more sub-strategies. Both games contain a set of *actions* $\{a_0, a_1, ..., a_n\}$ and a set of *resources* $\{r_0, r_1, ..., r_n\}$, that will be used to define the problem and the strategies. In both games the actions are dependent on the resources, meaning that actions require certain resources.

4.1 Simple game

Imagine a game where the goal is to produce oil (r_1) using oil refineries (a_0). The oil refineries need power plants (a_1) to supply for their energy (r_2) need. The oil decreases or increases over time based on the number of oil refineries (rs) that are built using action a_0, and the number of power plants (ps) that are built using action a_1. The game starts with a fixed amount of gold (r_0) and there are no actions that increase gold. One constraint is that none of the resources may completely deplete (all $r_i \geq 0$). In Table 2 the cost of the actions, in terms of resources, are defined:

	ΔGold (r_0)	ΔOil (r_1)	ΔEnergy (r_2)
Oil refinery (a_0)	-25	$+10(rs+1)-5ps$	-5
Power plant (a_1)	-25	$+10rs-5(ps+1)$	$+10$
Do nothing (a_2)	0	$+10rs-5ps$	0

Table 2: Action-resource table of the imaginary game

As can be seen in Table 2 gold and energy increase or decrease with a constant amount, while oil increases or decreases over time. In short, oil refineries cost gold and energy, and obtain oil. Power plants cost gold and oil, and generate energy. Action a_2 does not change the game state, other than the oil in-/decrement at each time step.

In this imaginary game the learner can interact by choosing one of the three actions (note that doing nothing is also a choice) during every time-step. Defining the problem in the next section is the first step of structuring the game.

4.1.1 Problem definition

Using the game described in the previous section and using the resource variables in Table 2, the problem definition of this game would look like:

- **Initial state**: 100 Gold (r_0), 10 Oil (r_1), and 0 Energy (r_2)
- **Actions**: Build an Oil Refinery (a_0), build a Power Plant (a_1), or do nothing (a_2)
- **Constraints**: The game state cannot have less than 0 Gold (r_0), less than 0 Oil (r_1), or less than 0 Energy (r_2).
- **Goal**: Produce more than 100 Oil (r_1)

The problem definition consists of the four components presented in Section 3.1. The goal is to choose a sequence of actions from the initial state to a goal state, producing more than 100 oil.

4.1.2 Strategies & Feedback

Using the problem definition in Section 4.1.1 some approaches to define a strategy will be explored. After the definition of a strategy, the construction of the feedback is discussed.

Strategies

To make sure that there actually is a sequence of actions that result in a goal state for this example problem, a tree with all possible states can be constructed. We can now traverse the tree with all possible states and look for a path that reaches a goal state. A resulting path, consisting of a sequence of actions, will look like:

$$\{a_1, a_0, a_0, a_2, a_2, a_2, a_2, a_2, a_2\}.$$

In other words, for every two oil refineries (a_0) the strategy should build one power plant (a_1). Since the actions a_0 and a_1 both cost 25 Gold (r_0), this sequence of three actions can only be performed once, since the sequence would cost 75 Gold. Recall that action a_2 means doing nothing resulting in the increment of Oil (r_1), since this is the only resource that decreases or increases over time.

Defining the strategy

To come up with an optimal strategy, we describe the strategy definition process. We first present three naive human attempts, second we describe an optimal strategy, and finally we propose a labeled strategy to provide feedback.

First attempt: The strategy should first check the validity of each action for the initial state ($r_0 : 100$, $r_1 : 10$, and $r_2 : 0$). Building an Oil Refinery (a_0) is not valid, since the energy would drop below zero. Building a Power Plant (a_1) is valid, since all resources stay above zero. Doing nothing (a_2) is also valid, since it will not alter the state. We can now randomly choose either a_1 or a_2. This way of defining the strategy is buggy since there is no indication if the goal will ever be reached and the information in the problem definition is not used.

Second attempt: The distance between the current state and the goal state could be used, to have an indication of the progression towards the goal. Since our goal is to produce 100 Oil (r_1), this would mean that the action that the learner should choose is the one that minimizes the distance between the current state and a goal state. Thus minimizing $|r_1(t+1) - r_1(t)|$, where t is the current time-step. This way an estimation of the progress towards the goal is obtained. If the knowledge of the prior attempt is used in combination with this observation, the learner would choose action a_2 from the valid actions ($a_1 \quad and \quad a_2$), since building a power plant (a_1) would increase the distance to the goal state. The strategy would never result in a goal state using this overly simple estimation.

Third attempt: Maybe a larger number of observations are required to solve this problem; Gold (r_0) is decreased by action a_0 and a_1 and is never increased. Oil (r_1) can be increased over time by a_0 and decreased over time by a_1. Energy (r_2) can be increased by a_1 and decreased by a_0. Furthermore doing nothing (a_2) will result in the increment or decrement of Oil (r_1), since this is the only resource that is linear with the time.

Step one: The action a_0 is the only action that approaches the goal state over time, so the main strategy is to perform this action. This action does decrease Gold and Energy with a constant amount, thus has no linear effect on Gold and Energy. Since a_0 can not be performed because the Energy would drop below zero, another action needs to be performed. The only action that increases Energy (which enables action a_0) is a_1. This action however decreases Gold by a constant amount and decreases Oil over time. The initial state has 10 Oil and 100 Gold, meaning that this action can actually be performed. The last action a_2 does not approach the goal state and does not increase Oil, Energy or Gold. Performing a_2 will never enable action a_0, thus the action a_1 is the only choice.

Step two: Action a_0 can now be performed which will decrease Gold and Energy by a constant amount, but will increase Oil over time. After performing action a_0 both Energy and Gold will meet the constraints. Action a_1 can also be performed once more, which will increase Energy, decrease Gold and decrease Oil over time. When a_2 would be performed Gold and Energy remain constant and Oil would decrease (since the action in step one will decrease Oil over time). Since the increment of Oil is the action that brings the current state closer to the goal, action a_0 should be performed.

In short, this third attempt tries to do an action that increases Oil (our first step). If this resource cannot be obtained using the set of actions, the strategy looks

for the resources needed to enable this action (our second step).

An optimal strategy: An optimal strategy would minimize the time to reach the goal, thus maximizing the Oil in the minimum number of time-steps. As can be observed in Table 2 at the beginning of this section, the set of resources are linear or constant over time. Using this information and the problem definition, a linear programming problem [48] can be defined. In this linear programming problem the actions $\{a_0, a_1, ..., a_n\}$ are the unknowns and the *resources* $\{r_0, r_1, ..., r_n\}$ are the known coefficients. Since each action can only be performed or not performed (and cannot be performed partially), the problem becomes an Integer Linear Programming (ILP) problem. Since the problem definition restricts the number of actions to one action per time-step, the problem becomes a 0-1 Integer Linear Programming Problem.

To define a linear programming problem using the problem definition in Section 4.1.1, the Oil needs to be maximized and the number of time-steps needs to be minimized. Since the objective function can only maximize one of these, the other needs to be fixed. When for example maximizing the Oil, the number of time-steps T are fixed. Although there may be a more computational efficient formulation for the problem, we solve the problem for every time T incrementally. Since this is not the main topic of this thesis, we leave the computational efficiency out of consideration.

A linear programming problem to solve the problem definition would look like (1). The first three constraints are the resource constraints for oil $O_{(t)}$, gold $G_{(t)}$, and energy $E_{(t)}$. The last two are to make sure that at most one action is performed every time-step. Note that if both a_0 and a_1 are zero, action a_2 is performed.

Maximize

$$O_t \quad (t = 0, 1, ..., T)$$

Subject to

$$\begin{aligned}
O_{(t)} : \sum_{s=0}^{t-1} 10(t-s)a_0(s) - 5(t-s)a_1(s) &\geq -10 && \forall t \\
G_{(t)} : \sum_{s=0}^{t-1} 25a_0(s) + 25a_1(s) &\leq 100 && \forall t \\
E_{(t)} : \sum_{s=0}^{t-1} -5a_0(s) + 10a_1(s) &\geq 0 && \forall t \\
a_0(t) + a_1(t) &\leq 1 && \forall t \\
a_0(t), a_1(t) &\in \{0, 1\} && \forall t
\end{aligned} \tag{1}$$

We try to maximize the oil O for a fixed T, subject to five constraints. The first constraint is the most complex, because oil decreases or increases over time. If we build an oil refinery at the start of the game, it yields more oil compared to building

it later during the game. This is why the first constraint subtracts the build-moment s from the time-step t.

The second and third constraint are less complex, since we just add all time-steps for both actions. The fourth constraint makes sure only one action can be performed per time-step. The fifth constraint defines the variables as binary variables.

The solution to this linear problem consists of the sequence of actions that should be performed for the shortest way to the goal, thus can be considered optimal. When this linear programming problem is solved using a linear solver (e.g. Gurobi [49]) the result is:

$$\{a_1, a_0, a_0, a_2, a_2, a_2, a_2, a_2, a_2\}.$$

If the linear programming problem becomes infeasible there is no solution for the problem, thus the goal can never be reached. In other words: If the problem is infeasible, no action can lead to the goal state, making it impossible to give hints. The linear programming problem only provides the optimal sequence of actions, but is less useful for detailed feedback.

Feedback oriented strategy: Our last strategy can give more detailed feedback using the strategy language by Heeren et al. mentioned in Section 3.3. For this game we define two sub-strategies, namely gaining oil and generating energy. To define a strategy for this game, the terms to be rewritten are a mapping of resources (the state). Particular actions can only be performed if their constraints on the state are satisfied. Performing an action possibly changes the state. When, for example, energy is low, the strategy should advise to build a power plant, instead of an oil refinery.

For every action the effect on the resources is defined in a rule. The resources consist of the three resources mentioned earlier and the number of oil refineries and power plants. The strategy contains the following five resources and three actions:

```
data Resources = R Oil Energy Gold OilRefineries PowerPlants
type Oil = Integer
type Energy = Integer
type Gold = Integer
type OilRefineries = Integer
type PowerPlants = Integer

a0 :: Rule Resources
a0 = describe "Perform a0" $ makeRule "a0" f
  where
    f :: Resources -> Maybe Resources
    f (R o e g rs ps) =
       Just (R (o + 10*(rs+1) - 5*ps) (e-5)    (g-25) (rs+1) (ps))

a1 :: Rule Resources
a1 = describe "Perform a1" $ makeRule "a1" f
  where
    f :: Resources -> Maybe Resources
    f (R o e g rs ps) =
       Just (R (o + 10*rs - 5*(ps+1)) (e+10) (g-25) (rs) (ps+1))
```

```
a2 :: Rule Resources
a2 = describe "Perform a2" $ makeRule "a2" f
  where
    f :: Resources -> Maybe Resources
    f (R o e g rs ps) =
       Just (R (o + 10*rs - 5*ps) (e) (g) (rs) (ps))
```

The strategy for this example game looks like:

```
main ::          Strategy Resources
main =           while (\(R o e g rs ps) -> o < 100) gainResources
gainResources = (check (\(R o e g rs ps) -> g ≥ 25) <*>
                  gainOil) ▷
                     (check (\(R o e g rs ps) -> g ≥ 50) <*>
                        gainEnergy) ▷ a2
gainOil =        check (\(R o e g rs ps) -> e ≥ 5) <*> a0
gainEnergy =     check (\(R o e g rs ps) -> o ≥ 5) <*> a1
```

The *main*-strategy checks whether the goal is reached, if not, more resources must be gained. The *gainResources*-strategy checks if there is enough gold to perform an action. If there is enough gold we first try to build an oil refinery, since gaining oil is the goal of the game. If building an oil refinery fails we try to build a power plant to gain energy. If both actions cannot be performed, the action a_2 (do nothing) is performed.

Note that the second resource check ($g \geq 50$) in the *gainResources*-strategy is to only build a power plant when an oil refinery can be built afterwards. Otherwise, building a power plant would slow down the oil production.

Now we can label the different sub-strategies to provide feedback in the form of hints. For readability we will leave out the lambda abstraction \(R r_0 r_1 r_2 ...) in the conditions of check. The labeled strategy now looks like:

```
main ::          LabeledStrategy Resources
main =           label "GoalCheck" $ while (o < 100) gainResources
gainResources = label "GoldCheckA" $
                  (check (g ≥ 25) <*> gainOil) ▷
                  label "GoldCheckB"
                     (check (g ≥  50) <*> gainEnergy) ▷
                        label "DoNothing" a2
gainOil =        label "ProduceOil" $
                  label "EnergyCheck" $ check (e ≥ 5) <*>
                     label "BuildOilRefinery" a0
gainEnergy =     label "ProduceEnergy" $
                  label "OilCheck" $ check (o ≥ 5) <*>
                     label "BuildPowerPlant" a1
```

Apart from generating hints, we also want to provide feedback about the action the learner choose. An example of an action that is not optimal is when the learner chooses action a_2 (do nothing) when there is enough gold to build an oil refinery (a_0). We now define buggy rules for a_0, a_1 and a_2, which contain the negation of the resource checks in the strategy:

```
a0g' :: Rule Resources
a0g' = describe "Buggy a0 gold" $ buggyRule "a0g'" f
  where
    f :: Resources -> Maybe Resources
    f (R o e g rs ps) = if g < 25 then
       Just (R (o + 10*(rs+1) - 5*ps) (e-5) (g-25) (rs+1) (ps))
       else Nothing

a0e' :: Rule Resources
a0e' = describe "Buggy a0 energy" $ buggyRule "a0e'" f
  where
    f :: Resources -> Maybe Resources
    f (R o e g rs ps) = if e < 5 then
       Just (R (o + 10*(rs+1) - 5*ps) (e-5) (g-25) (rs+1) (ps))
       else Nothing

a1g' :: Rule Resources
a1g' = describe "Buggy a1 gold" $ buggyRule "a1g'" f
  where
    f :: Resources -> Maybe Resources
    f (R o e g rs ps) = if g < 25 then
       Just (R (o + 10*rs - 5*(ps+1)) (e+10) (g-25) (rs) (ps+1))
       else Nothing

a1o' :: Rule Resources
a1o' = describe "Buggy a1 oil" $ buggyRule "a1o'" f
  where
    f :: Resources -> Maybe Resources
    f (R o e g rs ps) = if o < 5 then
       Just (R (o + 10*rs - 5*(ps+1)) (e+10) (g-25) (rs) (ps+1))
       else Nothing

a2g' :: Rule Resources
a2g' = describe "Buggy a2 gold" $ buggyRule "a2g'" f
  where
    f :: Resources -> Maybe Resources
    f (R o e g rs ps) =
       if ((e ≥ 5 && g ≥ 25) || (o ≥ 5 && g ≥ 50)) then
          Just (R (o + 10*rs - 5*ps) (e) (g) (rs) (ps))
          else Nothing
```

When the conditions in a buggy rule are met, we provide feedback to the learner about the chosen action. In the next section we describe how we are able to catch incorrect actions of the learner.

The optimal strategy, obtained from the linear programming problem, can be used to check if the action sequences of the strategies are equal. In an ideal situation a strategy definition should always give the optimal action sequence, that is given by the linear programming problem. In many situations, however, learning a strategy could be more important than the question whether the strategy is optimal.

4.1.3 Use case examples

We choose two use case examples based on two feedback questions [26]: 'How am I going?' and 'Where to next?'. The first example will advise an action (feed forward). The second example gives feedback on a performed action.

Feed Forward:
To imagine how the above defined strategy would provide feedback, think of a state with *oil*=5, *energy*= 0, *gold*= 50, a power plant and an oil refinery. The first strategy line checks if ($oil < 100$), which succeeds, and thus proceeds with the sub-strategy *gainResources*. This sub-strategy checks whether the constraint ($gold \geq 25$) succeeds, which is the case. Next, ($energy \geq 5$) fails and ($gold \geq 50$) succeeds, thus the sub-strategy *gainEnergy* is used. In this sub-strategy the comparison ($oil \geq 5$) succeeds. The labels corresponding to these checks and steps is:

```
["GoalCheck", "GoldCheckA", "ProduceOil", "EnergyCheck",
"GoldCheckB", "ProduceEnergy", "OilCheck", "BuildPowerPlant"].
```

These labels can be converted in feedback that can be presented to the learner, resulting in for example *'You did not yet reach the goal.'*, *'You have enough gold to build an oil refinery'*, and *'You need to produce oil, because you have too little oil.'*. If more detailed feedback is desired, the lines *'You need to produce energy, because you have too little energy.'*, *'You have enough gold to build a power plant.'*, and *'You have enough gold to build a power plant.'* can be added. If the feedback must be even more detailed, the line *'You should build a Power Plant.'* could be added. As can be seen in this use case, the feedback can consist of many levels of detail. The labels allow flexible feedback, since the selection of labels used to construct the feedback is adjustable.

Feedback:
Let us assume the learner selected action a_0 (build an oil refinery) in the same game state as above: (R 5 0 50 1 1). In this scenario the learner has already selected an action, and we want to give feedback about this action.

Every buggy rule is tested on the old game state (R 5 0 50 1 1) and the new game state (R 20 -5 25 2 1). With these game states rule a_0e' succeeds, thus action a_0 should not be used. This abstract form of feedback can result in *'You should not build an Oil Refinery, since you have too little energy.'*. To generate a hint for the learner we can now use the labels in the strategy.

With this simple game we try to explain how the approach can be applied. The game only contains a few resources and actions, but can, as can be read in the next section, be easily extended to multiple actions or resources. We use a strategy language to give feedback at any point in the strategy. To provide feedback to the learner we use labels, which enables us to provide feedback at various levels of detail. The feedback to the learner has a clear relationship with the problem-solving process, since the strategy and feedback are integrated in one language.

4.2 Game with multiple sub-strategies

The previous section described a simple game using only three actions and three resources. This section describes a more complex imaginary game on a more conceptual level. The game contains multiple sub-strategies and consists of seven resources and six actions. Figure 2 visualizes the relation between the actions and resources. The exact coefficients (or values) of the required resources are omitted, since the strategy is constructed in the same way as in the previous section. As can be seen in the diagram, every action requires one or two resources. The action *BuildHouses* requires *Isolation* and *Energy*, and gains *WarmedHouses*. This diagram will be used for a better understanding of the strategy definition in Section 4.2.2.

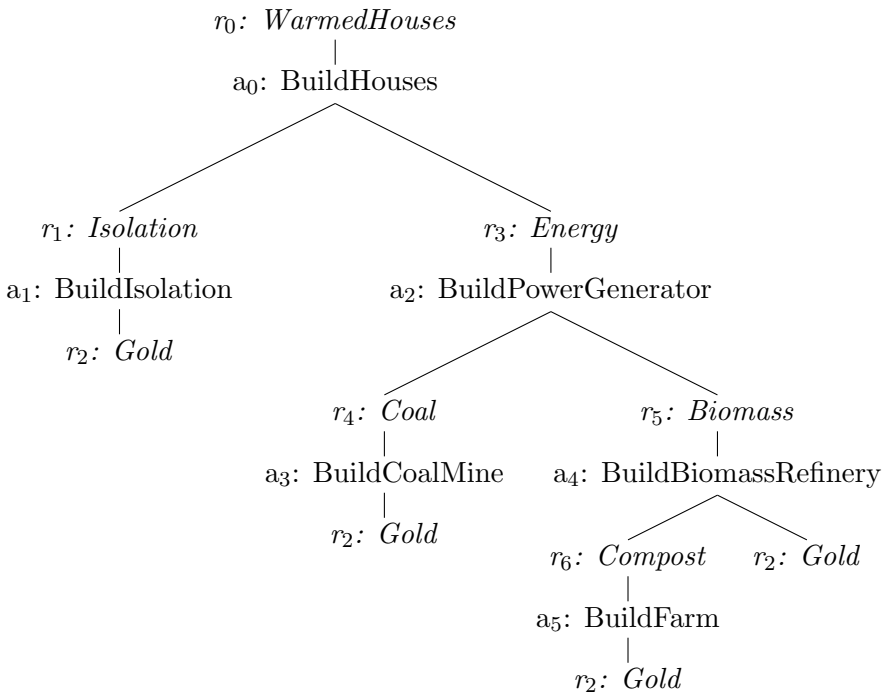

Figure 2: Conceptual representation of resources and actions

4.2.1 Problem

We define the problem of the second game as follows:

- **Initial state**: 0 WarmedHouses (r_0), 0 Isolation (r_1), x Gold (r_2), 0 Energy (r_3), 0 Coal (r_4), 0 Biomass (r_5) & 0 Compost (r_6)
- **Actions**: BuildHouses (a_0), BuildIsolation (a_1), BuildPowerGenerator (a_2), BuildCoalMine (a_3), BuildBiomassRefinery (a_4) & BuildFarm (a_5)
- **Constraints**: Resources can not be negative ($r_i \geq 0$)
- **Goal**: n WarmedHouses

4.2.2 Strategies & Feedback

The strategy in (4.2.2) is presented using the same language as in the former game. The strategy for this game consists of one main strategy and six sub-strategies.

```
main ::                  Strategy Resources
main =                   while(r0 ≥ n) buildHouses
buildHouses =            check (not (r1 ...)) <*> buildIsolation <||>
                           (check(not (r3 ...)) <*> buildPowerGenerator
                           ▷ a0)
buildIsolation =         check(r2 ...) <*> a1
buildPowerGenerator =    check(not (r4 ...)) <*> buildCoalMine <||>
                           check(not (r5 ...)) <*> buildBiomassRefinery
                           ▷ a2
buildCoalMine =          check(r2 ...) <*> a3
buildBiomassRefinery =   check(not (r6 ...)) <*> buildFarm <||>
                           check(not (r2 ...)) ▷ a4
buildFarm =              check(r2 ...) <*> a5
```

The structure of this strategy is visualized in Figure 2. The first strategy checks whether n or more houses are built, if not, houses should be built. The second strategy checks whether the constraints for building houses are met, if not, more isolation or energy is needed. To generate energy (fourth strategy) coal and biomass are needed. To acquire biomass gold and compost are needed. Compost can be obtained by building farms (last strategy).

Similar to the first example game, we label every part of the strategy:

```
main =                   while(r0 ≥ n) buildHouses

buildHouses =            label "BuildHouses" $
                         label "IsolationConstraint" $
                         check(not (r1 ...))
                           <*> buildIsolation <||>
                         label "EnergyConstraint"
                           (check(not (r3 ...))
                             <*> buildPowerGenerator ▷
                               label "HouseAction" a0)

buildIsolation =         label "BuildIsolation" $
                         label "GoldConstraintA" $
                           check(r2 ...)<*>
                             label "IsolationAction" a1

buildPowerGenerator =    label "BuildPowerGenerator" $
                         label "CoalContraint" $
                           check(not (r4 ...))
                             <*> buildCoalMine <||>
                               label "BiomassContraint"
                           (check(not (r5 ...))
                             <*>buildBiomassRefinery ▷
                               label "PowerGeneratorAction" a2)
```

```
buildCoalMine =          label "GoldContraintB" $
                            check(r2 ...)
                              <*> label "CoalMineAction" a3

buildBiomassRefinery =  label "BuildBiomassRefinery" $
                        label "CompostConstraint" $
                         check(not (r6 ...))
                           <*> buildFarm <||>
                         label "GoldConstraintC"
                            (check(not (r2 ...)) ▷
                               label "BiomassAction" a4)

buildFarm =              label "GoldContraintD" $
                            check(r2 ...)<*>
                               label "FarmAction" a5
```

We can define buggy rules, similar to the former game, to provide feedback on an action by the learner. Since this is trivial, we omit this.

This example shows a more extensive strategy structure, compared to the first example game. The strategy contains seven resources and six actions. Structuring the games using resources, actions, constraints, and a goal enables us to clearly structure a strategy. The structure of these games can easily be extended with more resources or actions, while the basic principle and structure of the strategy remains similar. We claim that the problem of this game can also be solved using linear programming, since the actions depend only on the resources.

5 Results

In this thesis we present an approach to provide feedback in resource management games using problem-solving strategies. We first define the problem of a game. The problem consists of an initial state, a set of actions, a model, and a goal constraint. We use this problem structure to describe a strategy that aims to solve the problem. The learning goal is to solve the problem of a game to get to a goal state. We define a strategy to generate feedback for the learner during his or her problem-solving process. The strategy is defined in a strategy language using combinators by Heeren et al. [42].

To show how the approach can be implemented, we give two example games. To show the broad employability of our approach, we show the problem and the strategy for both games.

The first game shows how a simple strategy can be constructed. The validity of an action in this strategy is determined by the presence of resources. To ensure the presence of the necessary resources for an action, we define constraints for every action. The actions should be goal directed, meaning that the goal of a sequence of actions is to eventually satisfy the goal constraint(s) of the problem. The actions defined in the problem have different effects on the game state, and are dependent on resources. We first define the main action that brings us closer to the goal constraint. Then we determine the actions that provide the resources needed to

enable this action. Using the dependencies of the actions, we divide the problem in different sub-problems that we solve in sub-strategies. The constructed strategy is labeled to provide feedback at any moment in the strategy. We show what the feedback or feed forward looks like using the labels in the strategy.

To show the versatility of the approach, we describe a second game with more resources and actions. In contrast to the first game, some of the actions depend on multiple resources.

In our approach the strategy and the feedback are integrated, to provide feedback at any location in the strategy at any moment in time. Given any state of the game we are able to provide feedback on an already performed action and feed forward (a *hint*) to advise the next step. We provide feedback based on violated constraints and feed forward based on the current game state. The feedback consists of a 'buggy' rule with a violated constraint, while feed forward consists of a list of labels with an advised action. When feedback on a performed action is requested, our approach can be used to indicate what constraint of the strategy is violated, and thus what part of the strategies the learner did or did not understand.

With our approach, it is possible to provide feedback on various levels of detail. The feedback consists of a sequence of labels that all describe a part of the strategy. The order of the label sequence is equivalent to the increasing detail of the feedback. Detailed feedback consists of more labels, whereas few labels result in more high level feedback. The sequence of labels can be now translated to information (e.g. text or audio) that the learner can use to improve his or her performance.

6 Conclusion

In this concluding section we will evaluate the contribution of our approach. We look back at the research question: *"How can we provide feedback based on problem-solving strategies using the input of a learner in resource management games?"* To answer this question we must analyze the different parts of the question. Our main goal was an approach that provides feedback in resource management games. The approach should guide in the problem solving process with feedback based on the input of a learner.

We use a strategy language to describe the strategy that solves a problem. We show that we are able to provide feedback at any moment in time. The feedback can be fed back to the learner in various levels of detail.

Section 1.4 mentions resource management games where the learner builds a city, learns about the survival of animals, controls a government, tries to manage a soccer team or learns about economics. Most of these games have a set of predefined tips, instead of specific feedback in various levels of detail at any moment in time.

These resource management games share the same structure as the two example games we presented. In these games the learner must manage resources, like natural resources, money or buildings, using actions to reach a certain goal. These games aim to teach an underlying mathematical problem, and have a game state and a model with resource constraints. Because of this structural similarity, we could

define a problem with an associated strategy for these games.

We believe that our approach is not limited to resource management games only. In many games (e.g. strategy games) a part of the game is managing resources. Our approach can provide feedback on the resource management for such games.

6.1 Future work

In this thesis we restrict ourselves to actions that have linear effects on the resources. An opportunity for future work is to implement our approach in games with more complex mathematical effects on resources. The games we present contain only a few resources and a discrete set of actions. Constructing a strategy for games with more dependencies between actions, or actions on a continuous scale is a challenge for future work.

The 'buggy'-rules and the strategy are separated in the games we describe in this thesis. The information of the constraints on the resources of the actions in the 'buggy'-rules and the strategy are, however, equivalent. Therefore, we expect that it is possible to derive the buggy rules from the strategy, which makes it redundant to define the 'buggy'-rules separately.

In addition to the feedback for an individual learner, this approach makes it possible to collect information about parts of a strategy that are misunderstood by a group of learners (e.g. a class). Future statistical analysis could further improve the strategies, and the analysis could be used to discuss the misunderstood parts of the strategy.

What we noticed during our literature research, is that many algorithms in the field of artificial intelligence already describe strategies that can be used to generate feedback in serious games. Artificial opponents in many serious games use a strategy which can also be well suited to be taught to a human learner.

References

[1] David Crookall. Serious games, debriefing, and simulation/gaming as a discipline. *Simulation & Gaming*, 41(6):898–920, 2010.

[2] Michael Zyda. From visual simulation to virtual reality to games. *Computer*, 38(9):25–32, 2005.

[3] Michele Confalonieri, Giovanni Guandalini, Mauro Da Lio, and Mariolino De Cecco. Force and touch make video games serious for dexterity rehabilitation. *Studies in Health, Technology & Informatics*, 177:139–144, 2012.

[4] Frank Delbressine, Annick Timmermans, Luuk Beursgens, Maaike de Jong, Alexander van Dam, David Verweij, Maikel Janssen, and Panos Markopoulos. Motivating arm-hand use for stroke patients by serious games. In *34th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, 2012.

[5] Stephanie Studenski, S Perera, E Hile, V Keller, J Spadola-Bogard, and J Garcia. Interactive video dance games for healthy older adults. *The journal of nutrition, health & aging*, 14(10):850–852, 2010.

[6] Atif Alamri, Mohammad Mehedi Hassan, M Anwar Hossain, Muhammad Al-Qurishi, Yousuf Aldukhayyil, and M Shamim Hossain. Evaluating the impact of a cloud-based serious game on obese people. *Computers in Human Behavior*, 30:468–475, 2014.

[7] R. Guarneri, M. Brivio, G. Andreoni, and M. Mazzola. Engaging teen-agers in the adoption of healthy lifestyles for the prevention of obesity and related co-morbidities: The approach of pegaso. *HEALTHINF 2014 - 7th International Conference on Health Informatics, Proceedings; Part of 7th International Joint Conference on Biomedical Engineering Systems and Technologies, BIOSTEC 2014*, pages 569–576, 2014.

[8] Samuel Benveniste, Pierre Jouvelot, and Renaud Péquignot. The minwii project: Renarcissization of patients suffering from alzheimers disease through video game-based music therapy. In *Entertainment Computing-ICEC 2010*, pages 79–90. Springer, 2010.

[9] Philippe H Robert, Alexandra König, Hélene Amieva, Sandrine Andrieu, Francois Bremond, Roger Bullock, Mathieu Ceccaldi, Bruno Dubois, Serge Gauthier, Paul-Ariel Kenigsberg, et al. Recommendations for the use of serious games in people with alzheimer's disease, related disorders and frailty. *Frontiers in aging neuroscience*, 6, 2014.

[10] Fernando Fernández-Aranda, Susana Jiménez-Murcia, Juan J Santamaría, Katarina Gunnard, Antonio Soto, Elias Kalapanidas, Richard GA Bults, Costas Davarakis, Todor Ganchev, Roser Granero, et al. Video games as a complementary therapy tool in mental disorders: Playmancer, a european multicentre study. *Journal of Mental Health*, 21(4):364–374, 2012.

[11] Brent Cowan, Hamed Sabri, Bill Kapralos, Mark Porte, David Backstein, Sayra Cristancho, and Adam Dubrowski. A serious game for total knee arthroplasty procedure, education and training. *Journal of CyberTherapy and Rehabilitation*, 3:285–98, 2010.

[12] Bernhard Maurer, Fabian Bergner, Peter Kober, and Rene Baumgartner. Improving rehabilitation process after total knee replacement surgery through visual feedback and enhanced communication in a serious game. In *Proceedings of the 30th ACM international conference on Design of communication*, pages 355–356. ACM, 2012.

[13] SR Chen, YH Lu, SF Yang, and YH Lin. A laparoscopic surgery simulator for hand-eye coordination training. In *The Eighth International Conference on Innovative Computing, Information and Control*, 2013.

[14] Thomas M Nosek, Mark Cohen, Anne Matthews, Klara Papp, Nancy Wolf, Gregg Wrenn, Andrew Sher, Kenneth Coulter, Jessica Martin, and Georgia L Wiesner. A serious gaming/immersion environment to teach clinical cancer genetics. *Studies in health technology and informatics*, 125:355–360, 2006.

[15] Natalie Harrold, Chek Tien Tan, Daniel Rosser, and Tuck Wah Leong. Copyme: a portable real-time feedback expression recognition game for children. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*, pages 1195–1200. ACM, 2014.

[16] Gwénaël Changeon, Delphine Graeff, Margarita Anastassova, and José Lozada. Tactile emotions: a vibrotactile tactile gamepad for transmitting emotional messages to children with autism. In *Haptics: Perception, Devices, Mobility, and Communication*, pages 79–90. Springer, 2012.

[17] D. Geelen, D. Keyson, S. Boess, and H. Brezet. Exploring the use of a game to stimulate energy saving in households. *Journal of Design Research*, 10(1-2):102–120, 2012.

[18] Brian Orland, Nilam Ram, Dean Lang, Kevin Houser, Nate Kling, and Michael Coccia. Saving energy in an office environment: A serious game intervention. *Energy and Buildings*, 74:43–52, 2014.

[19] Christoffer A Björkskog, Giulio Jacucci, Luciano Gamberini, Tatu Nieminen, Topi Mikkola, Carin Torstensson, and Massimo Bertoncini. Energylife: pervasive energy awareness for households. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing-Adjunct*, pages 361–362. ACM, 2010.

[20] Aikaterini Bourazeri, Jeremy Pitt, Pablo Almajano, Inmaculada Rodriguez, and Maite Lopez-Sanchez. Meet the meter: visualising smartgrids using self-organising electronic institutions and serious games. In *Self-Adaptive and Self-Organizing Systems Workshops (SASOW), 2012 IEEE Sixth International Conference on*, pages 145–150. IEEE, 2012.

[21] Petar Jercic, Philipp J Astor, Marc Thomas Philipp Adam, Olle Hilborn, Kristina Schaaff, Craig Lindley, C Sennersten, and J Eriksson. A serious game using physiological interfaces for emotion regulation training in the context of financial decision-making. In *ECIS*, page 207, 2012.

[22] Marco Greco and Gianluca Murgia. Improving negotiation skills through an online business game. In *Proceedings of the European Conference on Game Based Learning*, pages 97–104, 2007.

[23] Anna-Sofia Alklind Taylor, Per Backlund, and Lars Niklasson. The coaching cycle a coaching-by-gaming approach in serious games. *Simulation & Gaming*, 43(5):648–672, 2012.

[24] José Fernando M Silva, João Emílio Almeida, António Pereira, Rosaldo JF Rossetti, and António Leça Coelho. Preliminary experiments with eva-serious games virtual fire drill simulator. *arXiv preprint arXiv:1304.0726*, 2013.

[25] W Lewis Johnson, Hannes Högni Vilhjálmsson, and Stacy Marsella. Serious games for language learning: How much game, how much ai? In *AIED*, volume 125, pages 306–313, 2005.

[26] John Hattie and Helen Timperley. The power of feedback. *Review of educational research*, 77(1):81–112, 2007.

[27] Wendy Jaehnig and Matthew L Miller. Feedback types in programmed instruction: A systematic review. *The Psychological Record*, 57(2):219, 2007.

[28] I Dunwell, S De Freitas, and S Jarvis. Four-dimensional consideration of feedback in serious games. *Digital games and learning*, pages 42–62, 2011.

[29] Valerie J Shute. Focus on formative feedback. *Review of educational research*, 78(1):153–189, 2008.

[30] Roger Nkambou, Riichiro Mizoguchi, and Jacqueline Bourdeau. *Advances in intelligent tutoring systems*, volume 308. Springer Science & Business Media, 2010.

[31] Alla Anohina. Advances in intelligent tutoring systems: problem-solving modes and model of hints. *International Journal of Computers, Communications & Control*, 2(1):48–55, 2007.

[32] SM Alessi. The application of system dynamics modeling in elementary and secondary school curricula. In *RIBIE 2000-The Fifth Iberoamerican Conference on Informatics in Education*, 2000.

[33] Mary L Gick. Problem-solving strategies. *Educational psychologist*, 21(1-2):99–120, 1986.

[34] Monte Cristo. Cities xl. [Steam Download], 2009.

[35] Maxis. Simcity. [Origin Download], 2013.

[36] Enercities. http://www.enercities.eu/. Accessed: 2015.

[37] Papegaaien simulatie. http://www.betasimulaties.nl/sims/eco/. Accessed: 2015.

[38] Positech Games. Democracy 3. [Steam], 2013.

[39] Gamebasics. Online soccer manager. [Web], 2004.

[40] Sports Interactive. Football manager 2016. [SEGA Download], 2015.

[41] Kurt VanLehn. The behavior of tutoring systems. *Int. J. Artif. Intell. Ed.*, 16(3):227–265, 2006.

[42] Bastiaan Heeren, Johan Jeuring, and Alex Gerdes. Specifying rewrite strategies for interactive exercises. *Mathematics in Computer Science*, 3(3):349–370, 2010.

[43] Bastiaan Heeren, Johan Jeuring, Arthur Van Leeuwen, and Alex Gerdes. Specifying strategies for exercises. In *Intelligent Computer Mathematics*, pages 430–445. Springer, 2008.

[44] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Always learning. Pearson, 2014.

[45] Ideas tutorial. http://ideas.cs.uu.nl/tutorial/. Accessed: 2015.

[46] J v Neumann. Zur theorie der gesellschaftsspiele. *Mathematische Annalen*, 100(1):295–320, 1928.

[47] E. Rasmusen. *Games and Information: An Introduction to Game Theory*. Wiley, 2006.

[48] George Bernard Dantzig. *Linear programming and extensions*. Princeton university press, 1998.

[49] Gurobi. http://www.gurobi.com/. Accessed: 2015.