If {battleState = BattleState.standby}: Bringing the Gamer Into Play in Computer Game Development[1]

SANDRA PLONTKE

Computer games are frequently discussed with regard to those who play them, usually focusing on children and adolescents, especially young Western males. Within these discussions, we can identify different concerns that are associated with different problematisations, e.g. the question of harmful effects that computer games may have on the player. In Germany computer games, children, young adolescents and the topic of media harm are fairly often grouped in one single collection and thereby related to each other. As it becomes clear throughout the contributions of this book, those who regularly deal with questions about potential negative effects of computer games are among others anxious parents, legislators entrusted with the protection of minors, reviewers working for entertainment software self-regulation bodies like the German USK (cf. Schank, 2017); moreover, there are journalists, educators and psychologists interested in an possible link between computer games and the aggressive behaviour of children. They all share certain concerns regarding computer games and children, frequently addressing them through categories in which children as vulnerable subjects in need of protection are endangered by specific products of the entertainment industry. These concerns do also exist among the developers of computer games in Germany, where I did two years of ethnographic observations. After all, they cannot ignore the critical discourse about the potential harm their products may cause and for many reasons, including moral and economic considerations, they have to take a

1 I thank Pradeep Chakkarath for his valuable suggestions and linguistic support. I also appreciate the exchange I had with Daniel Theunissen about various topics that I deal with in this text.

stand. In Germany, this kind of pressure on the game development industry increased about twenty years ago when a couple of tragic shooting incidents triggered a broad debate about the deadly effects of so called *killer games*, i.e. computer games that require the player to kill human or human like figures in order to win[2]. Quickly, the producers of such games were caught in the crosshairs themselves and the smoke from the occasionally fiery debates is only slowly clearing. The industry continues to struggle with strong stigmatization, as the following quotes from game developers illustrate:

We were accused of breeding rampage killers and contributing to the brutalization of society. (Game designer)[3]

At some point, the only question that mattered was if the game industry fostered a culture of media usage that gets children addicted to computer games […]. I've spent a long time thinking about child protection, had sat a lot in podiums and had discussions with politicians […]. At that time […] I gathered fifty thousand signatures for a petition against the defamatory reporting on German public service television. (Representative of a game developer's marketing department)

2 The term *killer game* was probably introduced into the German political and popular discourse about computer games by Günther Beckstein, the former Bavarian Home Secretary. He used that expression when addressing a rampage that took place in Bad Reichenhall in 1999 and left five people dead, including the 16-year-old attacker. Subsequent school shootings committed by adolescents in Erfurt (2002), Emsdetten (2006) and Winnenden (2009) made *killer game* a catchword in debates about requirements for prohibition of certain computer games. The almost forgotten *killer game* debate was recalled to memory after an 18-year-old's rampage in Munich in 2016 when Thomas de Maizière, the Federal Home Secretary, in his press conference on 25 July, 2016 concluded that it is not to be doubted that the unbearable amount of violence-promoting games on the Internet has a detrimental effect on the development of young people and that no reasonable person can deny that (dbate, 2016). These debates did not come up with a clear or uniform definition of what *killer game* exactly means. It can be noticed, however, that most critics aimed particularly at so-called *first-person* and *third-person shooters*, marking them as games in which committing cruel and deadly acts of violence against human and other living beings contributes largely to the player's success.

3 Unless quoted differently, the quotes are taken from interviews that I conducted during my fieldwork in different developer studios and with freelancers of the industry. All translations into German were done by me.

All these journalists who are not so familiar with the matter, especially TV journalists, they always have to deliver pictures – and what do they do? Of course, they'll show you those six seconds of 15 hours of play which are particularly telegenic and help support their point: 'Oh, look here, what an orgy of violence that is!' And if some boneheads let their children play games that were released for players as of eighteen years of age, well – they should get a rap on the knuckles, you cannot control that. But, seriously, when do you have control at all? (2D artist)

These typical remarks from industry members reflect certain themes regarding computer games: media usage, media addiction, media violence, rampage, brutalization of society, age-appropriate game contents and control of children's access to games, portrayals of the game industry in the media and disputes between the industry, journalists and politicians. Developers of computer games, however, do not only refer to these issues in various debates, but take them up in their development practice and integrate them into the design of computer games. They do not produce games independently, not even if they are working on so-called *indie games*, i.e. independent computer games produced without the financial support, demands or interventions by a publisher. Game developers take measures that are always geared to prospective players or users who are inscribed into the games along with crucial aspects of the users' socio-material worlds (cf. Akrich, 1992). In the actions that developers take, they reflect genre conventions, politics, laws as well as historical and cultural ideas regarding games, players, entertainment and graphic representations, for example, representations of environments, gender, attractiveness, violence, war, heroism, etc.

"Designers", says Akrich (1992, p. 208) in her work on the de-scription of technical objects, "define actors with specific tastes, competences, motives, aspirations, political prejudices, and the rest" and try to inscribe all of these into a design. She describes *de-scription* as the process in which the artefact meets the user and his or her setting, the process of actual use in which the script embedded in the object unfolds *in situ*.

Against this background, the development of computer games can be regarded as a reflexive process in which the aforementioned infrastructural nodes, i.e. various concerns are anticipated. Finally, it is especially the technical infrastructures in which the game developers' practices and the design object, i.e. the computer game, unfold themselves: certain software programs for image processing such as *Photoshop* and 3D modelling software such as *Maya*, but also programming environments with specific programming languages and logics, as well as the Internet

and its mediators like Google and Facebook. All these different aspects are actively involved in the *con*figuration of computer games[4]. What game developers do and how they do it, the *modus operandi* of their actions, cannot simply be attributed to individual aesthetic preferences; it can rather be understood as a multifariously interwoven, relational practice. Computer games are socially configured and technology, including the technological aspects of a computer game's design, must anticipate different interests in order to solidify the games' infrastructures and thus the social aspects of the games themselves[5].

This is just a rough sketch of different references and arrangements in which the practices of game development are integrated and where they unfold themselves. It is in this complex network in which different concerns emerge with regard to the relationship between computer games and players. These are the concerns which in turn are actively involved in the development and shaping of computer games (concerning the story, game mechanics, graphics, etc.). I will try to illustrate some of these practices in the following; but first I would like to make a few brief remarks on a concept that I have already used a couple of times in this introduction − the concept of *concern*.

CONCERNS AS THE EFFECTS OF HETEROGENEOUS GATHERINGS

What I mean when I talk about concerns is by no means confined to problematisations, although my introductory remarks may have created that kind of impression. Against the background of my preliminary remarks, concerns can be understood as the effects of complex heterogeneous arrangements; they develop their respective modes of existence in situated relational practices and are in this sense

4 For an understanding of graphic design in game development as *heterogeneous engineering*, cf. Plontke (2016).

5 A boring game, for example, which does not meet the player's desire for entertainment and fun, may not be played for long. A game that is not developed according to the guidelines of the USK may not be published and is added to the Federal Review Board for Media Harmful to Minors' (BPjM) list of harmful media, the so-called *Index* (cf. Dreyer, 2018, this volume). A player will not respond to a game's graphic if it for instance does not meet the requirements of his or her wish for realism. The player will thus be annoyed and not immersed into the game's spell. Since game developers want "to keep the player in the game world as long as possible" (game designer), they need to be concerned with all these aspects.

practical tasks. In reference to Heidegger's critical and differentiated analysis of the concept of a *thing*[6] (Heidegger, 2006; 1986), Bruno Latour emphasizes this aspect of concerns when he states that they are assemblies, i.e. gatherings of things which transform an immutable neutral fact – a so-called *matter of fact* – into something that concerns us – a so-called *matter of concern*[7] (Latour, 2004; 2005; 2008a; 2008b). I would like to use the notion of concern similarly: Things are seen as assemblages, i.e. they congregate in a conglomerate of manifold relationships and are simultaneously constituted by them; concerns are seen as a result of these heterogeneous gatherings arising in and out of practices that we engage in when dealing with things[8].

6 Heidegger's (1927/2006) famous re-conceptualization of *Ding* (thing) is based upon an etymological analysis of the Old High German word *thing* which denoted a governing assembly of ancient Germanic tribes. The corresponding verb *dingen* (which is *thingan* in ancient forms of Anglo-Saxon languages) means the negotiation of a cause in an official assembly or in court. In his critical assessment of the degeneration of the word *thing* in the history of philosophy and contemporary everyday language, Heidegger points out that the concept was not originally used to denote a neutral or objective entity but rather to refer to *an issue at stake*, i.e. a matter of concern.

7 In Latour's (2004) essay *Why has critique run out of steam? From matters of fact to matters of concern*, later particularly in Latour's (2008a) *What is the style of matters of concern?*, he calls for a re-focusing of scientific research interested in the relationship between politics and science. He asks that the view of science be directed from the analysis of facts (*matters of facts*) to so-called *matters of concern*. Already in *Laboratory Studies* (Latour & Woolgar, 1986; cf. Latour, 1987) Latour deconstructed the objective appearance of scientific facts and exhibited their production: facts are not to be understood as objective, they are socially and politically co-constructed, their supposedly direct reference to reality is exposed as a chimera: the facts pretend to reflect reality by hiding the reference chains between reality and the facts in a black box. In this sense, objectivity is the result of *purification* as Latour (1993) called this process himself. In this consequence, he confronts the allegedly simple facts with the concept of the often interwoven *matters of concerns*. Here the term is to be used with regard to the social construction of technologies such as computer games or other technological phenomena (cf. Sørensen, 2018, this volume).

8 In the light of my brief remarks on this issue some readers may ask themselves: What would not be a matter of concern? Does the answer not depend on one's point of view? In a certain sense, yes. From a methodological point of view, the answer to the question of whether something is a matter of concern or a matter of fact appears as the result of

In this view, a computer game or a singular game asset, i.e. a single game content, is understood not as an object, i.e. as a matter of fact, but rather as a thing in the original etymological sense: it is seen as a gathering, a network that maintains a variety of relationships, as for example, relationships to people, opinions, ideas, interests, (world-)views, requirements, relationships to other things, the concerns of others, etc. It is these relationships that make them a *matter of concern*. As such, "they have to be explicitly recognized as a 'gathering', as a thing and not an object" (Latour, 2008a, p. 48). Elsewhere, Latour speaks of these gathered objects as "objects of design" (Latour, 2008b, p. 2), which corresponds to the idea of computer games as designed artefacts. Based upon another etymological recourse, Latour conceives of the act or process of designing as *drawing* and thereby draws our attention to an essential feature of the design process which consists of "drawing things together"[9]. In his own words:

> To think of artefacts in terms of design means conceiving of them less and less as modernist objects, and conceiving of them more and more as 'things'. To use my language artefacts are becoming conceivable as complex assemblies of contradictory issues (I remind you that this is the etymological meaning of the word 'thing' in English – as well as in other European languages). When things are taken as having been well or badly designed then they no longer appear as matters of fact. So as their appearance as matters of fact weakens, their place among the many matters of concern that are at issue is strengthened. (Latour, 2008b, p. 4)

In the following, I would like to look at design practices in game development and illustrate how things are drawn together in different situations. Drawing upon partial aspects of the programming and the development of the game mechanics for a combat system, I will try to show how the prospective player, his or her gaming experiences and the code for the combat system are brought together. Here, my focus will be on the question how the prospective player is brought into play, i.e. how he or she is represented in the code that the programmer writes – and what

---

a certain perspective, namely a relational view of things, one that Actor-Network Theory (ANT) and other praxeological approaches apply or even demand out of their theoretical convictions. It is precisely this change of perspective that Latour wants to establish: the replacement of focusing on facts and searching for their truth by a thorough analysis of how matters find their respective mode of existence in different socio-material arrangements. Through this refocusing we learn new things about matters. Thus, a matter of concern in methodological terms is always the effect of a perspective.

9 *Drawing things together* is also the title of an article written by Latour (1990).

kind or image of a player serves as a working model within a development practice which is concerned with aspects surrounding a specific game content like, e.g. a combat situation. Although the issue of combat, especially with regard to armed violence, is only one of many examples that could serve for illustrating my thoughts, one cannot ignore that it is one of the most discussed topics within debates about the potential effects that computer games might have on the players. Therefore, I will use the example that I selected in order to touch on that topic at least marginally. In attempting to draw all these strings together, the text will rather incidentally outline a minimal methodology for the study of code and programming practice.

THE COMBAT SYSTEM 2

Introduction to the Scenery and Some Methodological Considerations

The following episode stems from my ethnographic observations in the game industry and deals with the design and programming of a combat system for a computer game prototype.

I am looking over the shoulders of a young programmer I here call Sam. He is an intern in a game development studio which produces computer games and apps, mainly for children and young adults. Sam was asked to develop a combat system for a game prototype. While working on his task, he is documenting his work in a written internship report.

Sam is sitting in front of two computer screens and is doing various things. I am not considerably familiar with the complex art of programming but since much of what Sam does is visualized on the screens and also sketched out in his internship report, his work is at least partially observable and accessible to me. In the following let us concentrate on Sam's work by 1) focusing on the left-hand screen, the so-called *editor*, where Sam's program code for the *Combat System 2* is translated into a visualized 3D-animation. Let us then 2) take a look under the surface of these visualizations resp. graphical animations and for this purpose turn to the right screen where we can see the programming environment in which Sam develops the code for the *Combat System 2*. Here I will focus on a code segment which was written before the observation began, deconstruct it and try to reconstruct the

practices of programming and the emerging concerns regarding code and player[10]. In a next step 3) I will look at a short paragraph from Sam's internship report that documents the design of the code segment in question. The internship report can be seen as a kind of additional reflection of Sam's own programming work, i.e. as *reflection on action*, to put it in the words of the design scholar Donald Schön (1983). As Schmidt (2012) rightly observes, programming usually works without much talk. In addition to the taciturnity which characterizes the programming of the code and is interrupted only occasionally by interjections like "m-hm", "oh!", "huh?", "damn!", etc., it is especially the issue of visibility and invisibility that presents a challenge to ethnographic research. As Schmidt states, the difficulty is that programming, although it partially consists of observable actions, is also performed and subjectively experienced as an internal act of thinking; "*Denkhandeln*" (Schmidt, 2010; 2012, p. 287). Schmidt tries to meet this difficulty with micro-analytical descriptions that focus on the person's gestural and bodily performances and are supposed to make the alleged internal act of thinking *speak*. Since my focus will be on the concerns regarding code and player as they arise in the practices of programming, my approach is a different one, although it is confronted with similar methodological problems as mentioned by Schmidt. To me the code, its pictorial representation or translation in the editor as well as the excerpt from the internship report are certain articulations of an otherwise rather silent practice. I will use these elements as a resource for my following analytical descriptions. For our purpose they should help to get a better glimpse of programming practices and some of the concerns regarding the player that they convey.

By this means, programming can be made accessible and recognizable without adhering to the purely mentalistic idea of a mainly inner process[11]. Furthermore, I will refer to other ethnographic observations that illustrate the socio-material nature of programming and I will consider particular comments and explanations by

10 As Rob Kitchin shows in his discussion of six empirical approaches, the demands for a thorough research on algorithms are high: "[…] examining source code (both deconstructing code and producing genealogies of production); reflexively producing code; reverse engineering; interviewing designers and conducting ethnographies of coding teams; unpacking the wider socio-technical assemblage framing and supporting algorithms; and examining how algorithms do work in the world" (Kitchin, 2014, p. 3).

11 In particular, the *Ethnographies of Code* in TeamEthno (2006) and their perspective on the mundane practices of programming are a critique on cognitivist approaches to programming. See also Graham Button & Wes Sharrok (1995).

a second programmer, who I here call MacGuyver, and who was partly involved in the development of the code for the *Combat System 2*.

In Front of the Code == The Images

As already mentioned, in public discourse computer games are frequently adressed through problematizations, especially in relation to children who are seen as vulnerable subjects in need of protection. Particular game contents, e.g. combat scenarios in which characters are attacked, injured or killed, are discussed as "violent game contents" that could be considered harmful to the psychological development of minors. In our episode Sam has the task of developing a combat system. So, what is Sam concerned with when developing the *Combat System 2*? Let me pose this question more strikingly: Is he primarily concerned with developing a violent game content?

A look at the left screen (the editor) of Sam's working place may give reason to this assumption due to the fact that the animation shows a group of fighting characters: four figures (the heroes) are confronted with a huge opponent (the enemy). There is a series of attacks, defenses and counterattacks – a boss fight is going on. Everything still appears unfinished; the graphics and the animations are still in a draft, it is a prototype after all. And yet one might assume that – as a *matter of fact* – all this is about violence. After all, what I can see on the screen in front of me and with my own eyes is a fight – and is that not a matter of violence? "'Facts are facts are facts'? Yes, but they are also a lot of other things in *addition*" (Latour, 2005, p. 21). This statement by Latour will be taken into consideration in the following when we try to look behind the phenomena as they appear to us (e.g. on a screen).

Imagine worried parents entering their child's room and having a look at the computer screen while the offspring is fighting and killing on a battleground. In the eyes of the parents (but not necessarily in the eyes of the playing child) the scene on the computer screen might appear as violent. Their perception is ignited by watching the surface of the object that is mainly grasped through its aesthetic and symbolic aspects (cf. Latour, 2008b, p. 2). The following quote of a gamer gives an impression of such different or even discrepant point of views:

> You know, it [violence in computer games] has been quite an issue at home. My parents didn't appreciate me playing it [*Counterstrike*] but they didn't get it at all, they didn't understand why I was playing it. For them it was about me simply wanting to kill people or getting rid of aggressions [ ], but I always found that wrongful. (A gamer working at the quality assurance section of a game developer)

The deixis of images[12] – like all deictic expressions – is highly contextual. Iconic meaning does not show itself but is generated resp. unfolds in a network of image and viewer whereas the viewer's gaze is never innocent (cf. Sturken & Cartwright, 2001, p. 45). There is no "view from nowhere" (Nagel, 1986). But what is Sam's view? What is he focusing on?

As a programmer, Sam has his own perspective. He and other programmers are especially concerned with what lies beneath the screen and beneath the animated graphics of a computer game: Sam is mainly concerned with writing a code and employing certain rules. Even though the programmer's gaze oscillates between the two screens in front of him, between the code on the right and its visual display in the editor on the left, watching the editor screen primarily helps to get a *visual feedback* and thus to *control* the coding process: "Here, on that screen, that's the editor, I can see and control the code I've already written"[13] (Programmer).

At this point we can note that there are (at least) two sides at play, a kind of front and back resp. outside and inside of the (visual) artefact called computer game. Both sides, although they are interconnected, initiate different ways of seeing and meaning (cf. Latour, 2008b, p. 2). However, it has to be seen that the program code before us is not the direct undersurface of the outer pictorial appearance or visual outside but rather its mechanics[14]. Strictly speaking, the inner side or undersurface of the digital computer game image (3D graphics with their textures and colours) consists of a binary code that depicts the image as a surface phenomenon which brings the image into being[15]. Nonetheless, the mechanics

12 Cf. Gottfried Boehm (2007).

13 This can be understood as an example for what Charles Goodwin calls *professional vision*: "socially organized ways of seeing and understanding events that are answerable to the distinctive interests of a particular social group" (Goodwin, 1994, p. 606).

14 Because of its multi-dimensionality (audiovisuality, narrativeness, ludity and interactivity) it is, of course, a simplification to speak of a front and back of the artefact computer game. Even with regard to the iconic dimension of computer games, it is not easy to reduce it to a visible front or surface on the one hand and a technical backside on the other, since it would be necessary to differentiate between the still image, e.g. the 3D-model of an avatar and the same image as a moving image, i.e. the animated 3D-model of the avatar. The ontological analysis of the image into an upper and lower surface would thus be conceived in a sub-complex, almost essentialist sense. It is probably more appropriate to say that the constitution of the computer game image is enriched by other (technical) dimensions of the artefact, such as game mechanics and their code.

15 On the ambiguity of the digital image, see, among others, Frieder Nake (2005).

underlying the computer game images are closely linked to them, because they allow the visual worlds of a computer game to appear, to perform and to function in prestructured ways. Finally, the mechanics beneath the interface allow the player to interact with the visual worlds of a computer game in certain ways. Thus, particular events within the course of the play – and this means certain iconic formations – arise from game mechanics.

Behind the Image == The Code

On Sam's right screen we can see a few lines of code for the *Combat System 2* that he has already written (see Figure 1). As indicated before, the code presented below is only a part of the whole program code made available to me[16]. The process communicated in that code segment includes the order in which the combatants – consisting of four players (heroes) and the computer (enemy) – appear and act. In his internship report to which I will return below, Sam describes the code segment as "a sequence of interactions between the (respective) hero and the antagonist". So the code carries a specific sub-process of the entire battle, the right of attack, which is represented by the code segment and which has to be illustrated in the internship report. But let us take a closer look at that code segment to exemplify the idea of code as a logic-based set of rules:

16 This code is very much akin to so called pseudocode, a type of code that does not require a fixed formal programming language and grants individual design possibilities. It is particularly used for presentation and communication purposes, in our episode it is used in the context of the development of a computer game prototype and an internship report where it should outline a specific game sequence or process in its basic idea. Pseudocode thus serves in particular to reduce complexity and to illustrate a process (algorithm), whereby the program sequence can be represented independent of an underlying technology because of its freedom of form (cf. Mehlhorn & Sanders, 2008, pp. 26-31). At the same time, as another programmer explains, Sam's code already functions as source code that can be translated into machine language and then be performed. Thus, the code written by Sam is "a kind of mixture of source code and pseudocode" (programmer to whom I showed the code segment).

Figure 1: Code segment of the Combat System 2

```
if (battleState == BattleState.run)
      {
          // The first player is taken out of the queue
          currentCombatant = (Combatant) (combatQueue [0]);
          combatQueue.RemoveAt (0);

          // Defense
          if(currentCombatant.isDefense)
          {
              currentCombatant.isDefense = false;
              currentCombatant.restoreDefense ();
          }

          // Selection phase
          // Execute action
          if (currentCombatant.side == Side.enemy)
          {
              currentCombatant.randomAction (heroParty);
          }
          else
          {
              // Waiting for the input of the player/ Selection of action
              battleState = BattleState.standby;
          }

          // Final phase
          // Enqueue again
          combatQueue.Add (currentCombatant);
      }
```

Following the logics of Boolean algebra, the code consists of *if-/else-directives* and describes some of the rules (right of attack) that the combat action is based on. After the key word *if* we find an open curly brace followed by a formula block – a so called *if-block* – that describes certain conditions for the *if-directive*. In order to calculate and to perform the *if-block* resp. its conditions, the formula has to be provided with a Boolean *true* or *false* variable. When the *if-conditions* are "true" they will be performend; if they are "false" they will not be executed. If the conditions are "false" and therefore not met, an alternative "else" can be entered. In this perspective, code and thus the encoded combat action, appear as a step-by-step set of rules and conditions. Furthermore, the code is accompanied by comments introduced by "//". These comments should serve the reader – who in the case of a complex and complicated formula usually is another programmer – as a communication aid[17]. The comments may also help some readers of this text

17 Usually it is not necessary to use such comments in smaller teams where a more immediate way of communication (face-to-face) is common. Nevertheless, these comments

who are not familiar with the field of programming to decrypt the code segment. For the sake of clarity I will give a scriptual representation of the code segment presented above although the comments may already have done some of the translation work:

1) The first combatant is taken out of the queue.
2) If the fighter is in defense position (from a previous round), that status will be resolved.
3) If the current fighter is the enemy (enemy side), the computer randomly chooses an action (random action).
4) Otherwise it is waited for the input of the player who selects and executes an action (attack, special attack, defense, etc.).
5) The combatant is placed at the end of the queue again.

These descriptions of the single steps of a process represent a specific algorithm of the combat system: the order in which the involved combatants attack. So first of all, algorithms describe an operation, a process or a problem[18] as "a set of defined steps, [that] can be relatively easily codified; that is, turned into code that if executed will perform the algorithm" (Kitchin, 2014, p. 4; cf. Diakopoulos, 2013). In this sense, programming code can be seen as a translation work with the goal of solving a specific task; in Sam's case this task consists in translating the course and expiration of each combatant's right of attack into code. Kitchin refers to this aspect when he says that writing code consists of two central translations that are centered around producing algorithms: "First, translating a task or problem into a structured formula with an appropriate rule set [...]. Second translating

indicate that programming – against the stereotypical image of a nerd sitting lonely in front of his screen – is a social and collaborative practice and not the individual work of a single person. Often code is distributed, i.e. it is written, commented, discussed and explained. Single code segments resp. algorithms are separated, shared and modified by different programmers.

18 Algorithms as descriptions of processes or problems cannot only be executed by machines. We can find them in different forms and areas of our everyday life, e.g. as a recipe in a cookbook. In that sense Diakopoulos (2013, p. 3) states: "An algorithm can be defined as a series of steps undertaken in order to solve a particular problem or accomplish a defined outcome. Algorithms can be carried out by people, by nature, or by machines. The way you learned to do long division in grade school or the recipe you followed last night to cook dinner are examples of people executing algorithms".

this recipe into code that when compiled will perform the task or solve the problem" (Kitchin, 2014, p. 6).

This work of translation is one facet of the challenge the young programmer is faced with; he has to translate a specific task into a corresponding set of rules in such way that it can be compiled, that means that it can be translated into machine language in order to execute the algorithm.

Now let us return to a question posed before: How are computer games or parts of it, such as the game mechanics of a violent combat, configured through the practices of programming? In the light of the previous descriptions we may answer the question as follows: Within programming a combat is not associated with violence but is primarily configured logically resp. as a set of rules consisting of instructions and conditions. Thus, a combat or fight becomes a conglomeration of relational conditions, represented in the code's *if-* and *else-directives*. It will become clearer later on that the prospective gamer and his or her presumable future actions build a constitutive part within that logical structure. When asked what role the gamer plays in this code segment, another programmer explained to me: "The gamer is a kind of variable that determines which possible actions are executed" (MacGuyver, programmer). Here the gamer is referred to as a variable, i.e. he or she becomes technologically configured by fitting him or her into the structure of a Boolean thinking system. Reciprocally, the code itself is also shaped by the anticipated player and his or her actions. Thus, coding does not take place in a purely technical environment, in an isolated programming environment with a secret and hermetic programming language, but is socially configured by the presumed gamer and assumptions about his or her gaming experience and the acting style that he prefers. All of this must be inscribed into the design resp. into the code of the *Combat System 2*. In the code segment above, this aspect becomes clear in the commentary line where the player as an actor ("variable that determines which possible actions are executed") is explicitly mentioned in the code commentary: "//Waiting for the input of the player / Selection of action". However, in the lines of code themselves the player is not explicitly mentioned. Here the player comes into play through the *non-action* of his or her interaction partner, i.e. the computer in standby mode ("battleState = BattleState.standby"). At this blank state the player is expected to act. This situation is comparable to the turn-taking-machinery in a conversation: now it is the player's turn while the computer waits for the reaction, the choice of attack, special attack or defence and the corresponding input through the player's mouse or his or her keyboard. The way in which the code mobilizes the player's actions and his or her cognition gives us an idea of a distributed interplay in human-machine interaction (cf. Suchman, 2007). We can see that code is more than a rigid, deterministic set of rules, but gives the

player a scope of action, a freedom of choice within playing (I will return to this later). However, the presented passage of the code, the line that we called a *blank state*, can only be described as a *quasi-gap* because it is already filled with assumptions and expectations regarding a projected player, his playing behaviour and gaming experience (another aspect that I will return to). Finally, the code aims at orchestrating a particular way of acting and playing behaviour, one that culminates in a positive gaming experience[19] (cf. Jessen & Jessen, 2014). But what are the specifics of this way of acting and what kind of reasoning is connected to it regarding our episode? A look at Sam's internship report that deals with the code segment under consideration will shed some light on this question. I will focus on that report in the following in order to expand on the social configuration of the code and to better understand how the player, his reasoning, acting, gaming experience and the code design of the *Combat System 2* are *drawn together.*

Beside the Code == The Internship Report

As an intern Sam is considered a novice and has to document his programming practice and the design of the *Combat System 2* in an internship report. In ethnomethodological terms: As a young programmer Sam has to make his programming work, his decisions and the design of the combat system *accountable* (he has to be accountable to his supervisor who will evaluate the report and the design for the combat system in the end). Using this aspect methodically, I will put myself into the fairway of Sam's report and let it take my observations a little bit further. Sam's report may help to make programming work and the design of the *Combat System 2* more *recognizable*.

In the situatedness of playing, i.e. in the *gathering* of the computer game and the real player, the coded network of rules, the embedded restrictions and possibilities allow a new kind of player action to emerge: This new unfolding action is tactics. From the programmer's point of view, opening a door for tactics and strategic actions will channel a positive gaming experience. This is what Sam explains to me while writing down a short design document for the *Combat System 2* in his internship report: "It's all about the player being able to develop a strategy!" A look at this design document provides further information, including a comment on the code segment above:

19 During my fieldwork game developers told me again and again that their aim is creating games and that games should be fun. The fun aspect was also emphasized when the discussion touched upon the relationship between computer games and violence.

In this variant, a hero and an opponent act directly with each other, similar to rock-paper-scissors. This is to ensure that the player considers his turns very well before making his choice of action (attack, special attack, defence). The number of possible actions will also be limited in order to make the fight more tactical. (Sam's internship report)

Remember the *quasi-gap* that marked the gamer's turn and pushed him or her to select an action (attack, special attack or defense). This blank state mobilizes a cognitive component of the player. From Sam's report we learn that the code evokes a certain type of decision making. The programmed logic of the *Combat System 2* has to support the player's ability to act strategically: "that the player considers his turns very well before making his choice of action (attack, special attack, defence)". Furthermore, we get to know that regulated limitations of the player's actions ("the number of possible actions will also be limited") aim at making "the fight more tactical". Through these limitations (e.g. one may only attack three times and defend two times) the different options of action or abilities become a resource that can be used up and therefore must be handled prudently. As the report makes clear, it is a central goal to challenge the player cognitively. Instead of merely reacting according to coercive rules, the player has to develop his or her game or combat strategically and to make smart choices. But in order to make a choice, the mere opportunity of having a choice is not enough. The choice of a combat action must have a value and is associated with certain expectations regarding its effect which the player calculates within the game. Therefore, particular actions in a fight (defense, special attack, attack) are always linked to certain advantages and disadvantages that must be evaluated and carefully weighed by the player. This applies to every situation during the game and thus can be understood as a tool for strategy development[20].

In our case, Sam explains to me that the panel of available combat actions have certain attack value points and defense bonuses and therefore have to be used deliberately. Since the various combat actions and their values are not visible to me because they are neither elaborated in the intern report nor in the code segment

20 In his article on *The fundamental pillars of a combat system*, game designer Sébastien Lambottin (2012) also talks about "tools" regarding certain combat actions. He categorizes actions like *mellee-attack*, *normal shot*, *iron sight shot* or using a *grenade* as *a panel of abilities*. He writes: "Another way to think about the design of these abilities is to consider each one as a tool for the player" (ibid., p. 1). In the analytical perspective of the designer, weapons and violent actions such as a *mellee-attack* and the throwing of a hand grenade become a means to an end, to (neutral) tools of strategy development carrying a "risk versus reward trade-off" (ibid.).

provided to me (they are processed in a different class of code), MacGuyver gives me an illustrative example. In an e-mail exchange, he explains to me:

Consideration of the battle system is as follows:
1) I attack with a strong attack, but may be more vulnerable afterwards. (2 attack, 0 defense).
2) I attack normally and then have a normal defense value. (1 attack, 1 defense).
3) I defend and deal no damage for it, but get less damage myself. (0 attack, 2 defense).

As illustrated, each combat action has a certain attack and defense value – strong attack resp. special attack: "(2 attack, 0 defense)"; normal attack: "(1 attack, 1 defense)"; defense "(0 attack, 2 defense)" – that leads to certain consequences for the further course of the game. Whether the player defends or chooses a strong special attack goes along with certain advantages and disadvantages or rewards and risks which have to be taken into account in each situation.

I intervene: In this context, fighting in a computer game primarily becomes a question of strategical reasoning and acting – it is not about violence or killing as it might appear at first glance, for example when watching the editor, where the combat action is visualized on the screen; the surface of the code so to speak. However, within programming, the programmer is concerned with giving the player the rules at hand he needs to play a smart and thus joyful gameplay. As Sébastien Lambottin, senior game designer at Ubisoft Montreal, writes:

The main objective we have in mind when we design the gameplay mechanics of a combat system is to push the player to make clever choices and use the right ability at the right time. We want the player to be able to anticipate the next action he'll perform and also to develop a tactical plan during the combat [...]. So basically we want a system with multiple choices, but in which the player has to evaluate and choose the best option for each situation. (Lambottin, 2012, p. 1)

When Sam compares the strategy orientation of his code with "rock-paper-scissors", his report illustrates these aspects once more, because the popular children's game is not merely a game of chance but requires strategic and prospective thinking. It is a game that challenges the player's ability to assess and anticipate his or her opponent's actions and therefore demands the evaluation of the opponent's possible next turn as well as one's own options.

Again it is MacGuyver, the more experienced senior programmer of the studio, who gives me an explanatory example by putting himself in the role of the player in an imagined battle scenario:

Well, you try to judge your opponent, of course. For example, I have blocked two rounds in a row and the opponent can deal little or no damage with a normal attack. So I assume he chooses a strong [special attack] to break my defence, so I lower the shield and try to do more damage to him with a normal attack than he does to me (MacGuyver, programmer).

Because of a shared underlying principle even a children's game like *rock-paper-scissors* can be easily compared with a combat system for a computer game. In this section, it should become obvious that within the context of programming – as articulated in the code segment and in the internship report – the prospective player is configured as a tactician and fighting is conceived as a matter of strategic thinking and acting. Therefore, the programmer's concern is about providing the anticipated player with certain possibilities and abilities by means of programmed rules in order to make him or her act strategically. In this sense, algorithms effect something, or as Goffey (2008, p.17) puts it: "Algorithms do things, and their syntax embodies a command structure to enable this to happen". On the one hand, the code assigns the player to a specific role with a particular behaviour; the code affects and thus *con*figures the player. On the other hand, the programmed code does not represent a purely technical entity insofar as it integrates cognitive aspects that are mobilized in the act of *de-scription* (Akrich, 1992); the code is affected by the prospective player, it is socially *con*figured.

TRANSLATING CODE INTO A POSITIVE GAMING EXPERIENCE

Differentiating Player Mentalities

As I mentioned earlier, the set of rules laid down in the code is one part of the game mechanics that is constitutive for the player's interaction with the game. Thus, game mechanics are constructs of rules providing game play and gaming experience. Against this background, gaming experience is to be understood as a practical effect of the assembly of the player and the (rules of the) game[21]. Finally, a combat system that evokes tactical action and thinking can be seen as a mediator or, as Lambottin puts it, as a "tool" for such a positive gaming experience: "[O]ne

21 Obviously the code of the game mechanics is only *one* constitutive part of the gaming experience among others. In addition, it is also the graphics, the sound and the story of a computer game which are co-constitutive for the gaming experience of the individual player.

of the most engaging feelings a player can experience with a computer game is to feel smart and proud of his or her cleverness. And a combat system is a great tool to let the player experience this feeling" (Lambottin, 2012, p. 3).

However, in order to ensure a positive gaming experience, the design of the combat system must also consider a wide range of options and weigh up how the player may choose to play in specific situations. This poses a particular challenge for developers of computer games, since a standard player does not exist, but has to be generalized or formalized in order to be included in the code. For example, not each individual player follows the same tactics in a fight, nor develops the same preference for certain weapons or takes the same risk in a certain combat situation. Some players enjoy collecting points and things, they love exploring the game world and running through different levels without seeking confrontation with an opponent. On the other side, we also find players who are just looking for confrontation and fighting and try everything to kill the enemy. Therefore, computer game developers have to keep different types of players with different personalities in mind; players with different motivations and preferences that must be integrated into the game design[22]. These differentiations into different types of players must also be taken into account when programming. Regarding the *Combat System 2*, MacGuyver explains to me:

> One tries to make certain tactics possible through the rules, so that the player can take risks but does not have to. This way you can reach more players. If someone wants to play for safety, you want to allow this with rules as well so that the player has fun, a sense of achievement and a positive gaming experience. Forcing the player doesn't make him happy!

From this quote we learn that the rules of the combat system should not only support the player's tactical behaviour, but have to be differentiated by anticipating various, for example, risk-taking and risk-averse player types ("If someone wants to play for safety, you want to allow this with rules as well"). Last but not least, this differentiation has to be seen against the background of economic interests the gaming industry pursues by expanding the target group – another matter of concern in game development. Design and programming practices resp. certain codes or algorithms of computer games are closely related to economic factors. However, as we also learn from the quote above, programming is about creating "a positive gaming experience". The rules manifested in the code have to ensure that

22 The most prominent differentiation is probably the classification in player types according to Richard Bartle (1996): *Achiever*, *Explorer*, *Socializer* and *Killer*.

the player experiences "fun"[23] and "a sense of achievement" according to individual gaming preferences that the code structures.

Regulated Freedom

At this point I would like to return to the earlier mentioned idea of granting the player a room for manoeuvre or rather a certain freedom of action within a given framework of coded rules in order to foster a positive gaming experience. Whether this succeeds, however, is not only due to the different motivations and preferences of different player identities inscribed in the code, but especially to the fact that the rules evoking a positive gaming experience are not experienced as constraints. Certain rules evoking a positive gaming experience are not experienced as constraint; or as the programmer puts it: "Forcing the player doesn't make him happy!" It is the experience of freedom within playing that forms another important aspect of a positive gaming experience. As Johan Huizinga stated in his classical work *Homo ludens*, "play is a voluntary activity or occupation executed within certain fixed limits of time and place, *according to rules freely accepted but absolutely binding*" (Huizinga, 1949, p. 28, emphasis added). In this perspective, the code becomes a hidden actor, doing a kind of *invisible work* (Star, 1999) while becoming invisible itself. When the game is played, the code is translated into an experience and invisible as a determining set of rules. The translational achievement of the programmer does not only include converting a process into rules or an algorithm into code, but also designing it in such a way that it is transformed into a positive experience in the broader context of interaction with the player. Translations always lead to new formations, new figurations, folding and unfolding of (technical) objects and other entities, here the code for the *Combat System 2*[24].

Feeling Code

Ultimately, the correct balance determines whether the game ends in a positive gaming experience or not. To this regard, the "tool" (i.e. the combat system) has

23 According to Marc LeBlanc, who has developed a taxonomy of gaming fun, there are (at least) eight fun factors: sensation, fantasy, narrative, challenge, fellowship, discovery, expression and submission. In 2000 LeBlanc presented his ideas at the Game Developers Conference in San Francisco. His thoughts are written down in: Hunicke, LeBlanc & Zubek (2014). Cf. Salen and Zimmerman (2003, pp. 328-362).

24 For the concept of *translation*, see Callon (1986, p. 203).

to be adjusted appropriately. The right balance also depends on the fairness of the game: all players must have the same strengths (defence and attack values) and resources and must have equal chances of winning or losing. Too many challenges and, as a consequence, too many defeats cause frustration for players, the game is perceived as unfair, whereas too little challenge and too easy coping with the game leads to boredom. Therefore, the right balance has to be found and translated into the code. But how do you find this balance? MacGuyver gives the following explanation: "Of course you can calculate the fighting system and see if it's fair, but just because it's fair doesn't make it feel fair". Fairness is not just a question of mathematically correct computation, a matter of fact, but is presented as a kind of reception effect, a feeling of the player. Fairness thus becomes rather a matter of concern or – as I would like to call it – a *matter of act*. Something that is arithmetically and formally fair does not necessarily have to feel fair in gameplay, i.e. in actual game practice. Here, the correspondence between code (as a technical entity) and human gaming experience is questioned.

The challenge for the programmers now is to provide a translation that bridges and closes this gap between the code and the actual gaming experience which ultimately gives the code its reality. Akrich suggests oscillating between the inscribed and real world, albeit for the explorer of technology and design practices:

> Thus, if we are interested in technical objects and not in chimerae, we cannot be satisfied methodologically with the designer's or user's point of view alone. Instead we have to go back and forth continually between the designer and the user, between the designer's projected user and the real user, between the world inscribed in the object and the world described by its displacement. (Akrich, 1992, pp. 208-209)

This methodological conclusion drawn by Akrich that requires an oscillation between the world inscribed and the real world, does not only matter for (social) scientists, but is also guiding designers and developers of technologies such as programmers of computer games. Regarding the latter, this oscillation can be considered as an (ethno-)method to provide transfer and translation services between these worlds, between technology and experience. Sam and MacGuyver, for example, take on the role of the player and the situation of the game to add a real dimension to the projected player's experience. They move between worlds, between inscription and de-scription. Using small wooden figures and tokens they imitate the combat action and the game mechanics; the code is translated into a materialized game scenario of wooden figures, stones and cubes (see Figure 2).

Figure 2: Materialized game scenario

The defence and attack points embodied by the wooden pieces are varied and the respective victories and defeats are noted on paper. Your own gaming experience, "what it feels like", Sam says, becomes the starting point for the further development of the code for *Combat System 2* and flows into the programming practice. In the words of the programmers:

> We played this to see if the balancing is correct, for example, if I bet two of four life points and beat my opponent who has a certain amount of points and then take three points, do I always win or lose as a player? And how does that feel, even if you vary it? [...] I'm sure you could calculate it, but that's faster and you get a first feeling for whether it's fun or not.

At this point another facet of the material dimension of programming practices becomes visible. Programming is not to be understood as a purely mental or technical matter and is not a unidirectional process without detours. Consequently, algorithms or code cannot be seen as "purely formal beings of reason" as Goffey (2008, p. 16) states, even if this is how they are frequently presented by members of the technology and computer domain. Programming practices are embedded in

complex socio-material arrangements in which certain concerns regarding the relationship between computer game and players emerge.

CONCLUSION

Although the range of different computer games is almost unmanageably large, the discourse on computer games is often dominated by problematisations such as the question of the harmful effects of violent computer games on children and adolescents. We have used this observation as a starting point to take a more differentiated look at the question of what matters to the developers of computer games when they produce their products. As we said at the outset, computer game manufacturers are well aware of the social problematisations that their products give rise to, for example in political debates, in media coverage and in conversations between parents and their children. Developers of computer games are not simply aware of the ongoing debates but they also take the issues up in their development practice and even integrate some of them into the design of computer games. We have used this additional observation as a starting point to take a closer look at how this happens, based on a very narrow section of the production process of a combat sequence in a computer game. Based on these additional observations in the studio of a computer game developer, we were able to see how closely connected the so-called real world is with the so-called virtual world and how necessary it is for the actors in the computer game industry to re-establish and maintain this connection in their work. As our multi-fold analysis of programming practices showed, they are embedded in multi-layered socio-material arrangements and thus serve as valuable sources for the investigation of the relationship between technology and the social world. For example, we were able to see how developers anticipate and inscribe psychological traits (strategic thinking, fun, security, risk, sense of justice and fairness) into the code, which can then unfold in the game in a specific way through and with the player's actions. While the mechanics and rules offer freedom and are not simply constraining, the player's gaming experience is not purely subjective but also technically configured: players and the (coded) game are co-constitutive.

The methodological challenges for appropriate research into these interrelationships could only be addressed marginally. Nevertheless, I hope that I have at least roughly outlined what a more complex ethnographic approach to these challenges could look like. It will necessarily depend on a change of perspective that draws our main attention from matters of facts and shifts it to a sounder under-

standing of how matters, as for example computer games, find their mode of existence in varying socio-material arrangements that they simultaneously co-constitute.

It is within these arrangements where heterogeneous gatherings take place, where algorithms and players meet, where alleged matters of fact can be identified and analysed as matters of concern. As we could see, code is not a neutral and untouchable fact but is a matter of concern right from the start. It can be well or badly programmed, it can provide entertainment and enjoyment for the player or cause boredom and frustration; it can reach the player or make him leave – and as a consequence, these possibilities become an economic matter for game developers while they are a matter of entertainment for players. Of course, code can become a far-reaching problem and lead to moral questioning. The programmed set of rules could encourage or trigger certain questionable attitudes and behaviours, for example, if it tempts the player to prefer brutal over gentle problem solving or simply to spend money to get to the next levels. Against the background of what I have outlined in this text, the potential problems that appear here are always to be understood as social problems, sometimes serious social problems. However, in the light of what I have said, we should also be cautious about jumping to conclusions. Fighting and combat is not only violent and not only strategic, but much more in addition, it is both and more.

LITERATURE

Akrich, M. (1992). The De-Scription of technical objects. In W. E. Bijker & J. Law (Eds), *Shaping technology / Building society. Studies in sociotechnical change* (pp. 205-224). Cambridge, MA: The MIT Press.

Bartle, R. (1996). *Hearts, clubs, diamonds, spades: Players who suit MUDs*. Retrieved from http://mud.co.uk/richard/hcds.htm#1

Boehm, G. (2007). *Wie Bilder Sinn erzeugen. Die Macht des Zeigens* [How images generate meaning. The power of indicating]. Berlin: Berlin University Press.

Button, G. & Sharrok, W. (1995). The mundane work of writing and reading computer programs. In P. ten Have & G. Psathas (Eds), *Situated order. Studies in the social organization of talk and embodied activities* (pp. 231-258). Washington, D.C.: University Press of America.

Callon, M. (1986). Some elements of a sociology of translation: Domestication of the scallops and the fishermen of Saint Brieuc Bay. In J. Law (Ed.), *Power, action and belief: A new sociology of knowledge?* (pp.196-233). London: Routledge.

dbate. (2016, 25 July). *De Maizière gibt Killerspielen Mitschuld an Amoklauf in München (dbate)* [De Maizière puts part of the blame regarding the shooting rampage in Munich on killer games (dbate)] [Video file]. Retrieved from https://www.youtube.com/watch?v=Aa6EimWipic

Diakopoulos, N. (2013, 12 February). Algorithmic accountability reporting: On the investigation of black boxes [Web blog post]. Retrieved from http://towcenter.org/algorithmic-accountability-2/

Dreyer, S. (2018). The legal framework for computer games and child protection in Germany. In E. Sørensen (Ed.), *Cultures of computer game concerns:* The child *across families, law, science and industry* (pp. 95-112). Bielefeld: transcript.

Goffey, A. (2008). Algorithm. In M. Fuller (Ed.), *Software studies – A lexicon* (pp. 15-20). Cambridge, MA: MIT Press. doi:10.7551/mitpress/9780262062749.003.0002

Goodwin, C. (1994). Professional vision. *American Anthropologist, 96*(3), 606-633. doi:10.1525/aa.1994.96.3.02a00100

Heidegger, M. (1986). *Der Ursprung des Kunstwerks* [The origin of the work of art]. Leipzig: Reclam.

Heidegger, M. (2006). *Sein und Zeit* [Being and time] (19th ed.). Tübingen: Niemeyer.

Huizinga, J. (1949). *Homo ludens: A study of the play-element in culture*. London: Routledge.

Hunicke, R., LeBlanc, M. & Zubek, R. (2014). *MDA: A formal approach to game design and game research.* doi:10.1.1.79.4561

Jessen, J. D. & Jessen, C. (2014). Games as actors – Interaction, play, design, and actor network theory. *International Journal on Advances in Intelligent Systems, 7*(3-4), 412-422.

Kitchin, R. (2014). Thinking critically about and researching algorithms. *The Programmable City Working Paper 5.* doi:10.1080/1369118X.2016.1154087

Lambottin, S. (2012, August 15). *The fundamental pillars of a combat system*. Retrieved from http://www.gamasutra.com/view/feature/175950/the_fundamental_pillars_of_a_.php

Latour, B. (1987). *Science in action. How to follow scientists and engineers through society*. Cambridge, MA: Harvard University Press.

Latour, B. (1990). Drawing things together. In M. Lynch & S. Woolgar (Eds), *Representation in scientific activity* (pp. 19-68). Cambridge, MA: MIT Press.

Latour, B. (1993). *We have never been modern*. Cambridge, MA: Harvard University Press.

Latour, B. (2004). Why has critique run out of steam? From matters of fact to matters of concern. *Critical Inquiry – Special issue on the Future of Critique, 30*(2), 225-248. doi:10.1086/421123

Latour, B. (2005). From Realpolitik to Dingpolitik or How to make things public. In B. Latour & P. Weibel (Eds), *Making things public. Atmospheres of democracy* (pp. 14-41). Cambridge, MA: MIT Press.

Latour, B. (2008a). *What is the style of matters of concern? Two lectures in empirical philosophy*. Amsterdam: Royal van Gorcum.

Latour, B. (2008b). A cautious prometheus? A few steps toward a philosophy of design (with special attention to Peter Sloterdijk). In F. Hackney, J. Glynne & V. Minto (Eds), *Networks of design. Proceedings of the annual international conference of the Design History Society* (pp. 2-10). Cornwall, UK: Universal Publishers.

Latour, B. & Woolgar, S. (1986). Laboratory life: The social construction of scientific facts. Princeton, NJ: Princeton University Press.

Mehlhorn, K. & Sanders, P. (2008). *Algorithms and data structures. The basic toolbox*. Berlin: Springer. doi:10.1007/978-03-540-779780

Nagel, T. (1986). *The view from nowhere*. New York City: Oxford University Press.

Nake, F. (2005). Das doppelte Bild [The double picture]. *Bildwelten des Wissens. Kunsthistorisches Jahrbuch für Bildkritik, 3*(2), 40-50.

Plontke, S. (2016). How things are designed and how they design. In A. Bruni, L. L. Parolin & C. Schubert (Eds), *Designing technology, work, organizations and vice versa* (pp. 247-271). Wilmington, DE: Vernon Press.

Salen, K. & Zimmerman, E. (2003). *Rules of play. Game design fundamentals.* Cambridge, MA: MIT Press.

Schank, J. (2017). *W/wissen in der Alterskennzeichnung von Computerspielen. Eine praxeographische Analyse ausgewählter Entscheidungstexte aus dem Freigabeverfahren bei der Unterhaltungssoftware Selbstkontrolle (USK)* [Knowledge and knowing in age-rating computer games. A praxeographic analysis of selected written decisions from the ratings procedure at the Entertainment Software Self-Control (USK)]. Bochum: Westdeutscher Universitätsverlag.

Schmidt, R. (2010). Mikrogesten und die Beobachtbarkeit von Denkhandeln [Micro gestures and the observability of thinking as action]. In C. Wulf & E. Fischer-Lichte (Eds), *Gesten – Inszenierung – Aufführung – Praxis* (pp. 327-334). München: Wilhelm Fink Verlag.

Schmidt, R. (2012). *Soziologie der Praktiken. Konzeptionelle Studien und empirische Analysen* [Sociology of practices. Conceptual studies and empirical analyses]. Berlin: Suhrkamp-Verlag.

Schön, D. (1983). *The reflective practitioner: How professionals think in action*. New York City: Basic Books.

Star, S. L. & Strauss, A. (1999). Layers of silence, arenas of voice: The ecology of visible and invisible work. *Computer Supported Cooperative Work (CSCW)*, *8*(1-2), 9-30. doi:10.1023/A:1008651105359

Sturken, M. & Cartwright, L. (2001). *Practices of looking. An introduction to visual culture*. New York City: Oxford University Press.

Suchman, L. (2007). *Human-machine reconfigurations: Plans and situated actions.* New York City: Cambridge University Press.

TeamEthno-online (2006). *TeamEthno-online Journal, 2*. Retrieved from: https://archive.cs.st-andrews.ac.uk/STSE-Handbook/Other/Team%20Ethno/TeamEthno.html