8.

Source Code and Formal Analysis

A Reading of Passage

Ea Christina Willumsen

Transactions of the Digital Games Research Association
2017, Vol. 3, No. 2, pp. 213-235
ISSN 2328-9422
http://todigra.org

ABSTRACT

Analysis of the source code of video games is not an integrated part of the formal analysis. Rather, few scholars have investigated how an analysis of the source code can inform a hermeneutic reading of the game. In this paper, I will present a reading of the source code of *Passage* (Rohrer, 2007), argue for why a traditional focus on authorial intention is unnecessary when investigating the symbolism and metaphors of a game, and illustrate how source code analysis can inform the formal analysis of the executed game. Finally, I will discuss how the source code relates to the game as a 'work', and how it can be used

for studies of symbolism and metaphors. Thus, I will conclude that it is indeed a valuable method for game studies, although further studies should expand on the textual relation between executed game and source code.

Keywords

Source code, formal analysis, authorship, authorial intent

INTRODUCTION

The computational aspect of games has received little attention within game studies, and only some within software studies, where white-box analysis, the study of the source code, is not understood in relation to a hermeneutic interpretation. Hardware, code, and execution are commonly treated as a “black box”, where the scholar has no access to the actual code. Already in 1985, Buckles looked into the various computer files of *Colossal Cave Adventure* (Crowther & Woods, 1977), in order to quote all possible texts generated by the system in reply to player input. One of the first scholars to point out the relevance of reading the code of a game is Konzack (2002), who, in his otherwise very humanities focused seven layer analysis model includes both hardware and program code. He argues that “[i]n a complete analysis of a computer game every layer of the computer game in question should be analysed, but it is still possible to make an analysis of a computer game with out [sic] taking every layer into account” (Konzack, 2002, p. 92). Even though he does not use the term, this makes Konzack not only one of the first game scholars to argue for the relevance of studying software in game studies, where the source code is considered in relation to the executed game, but also one of the few, if not only, to state that a complete analysis must also include an analysis of the code.

There are two main reasons why the source code is not often considered in the formal game analysis: 1) the fact that we rarely have access to the source code of the game, and 2) that the scholar may not be able to translate the code into anything meaningful, which can be both due to lack of experience in reading code, or because the analysis of the program as static text does not contribute to the comprehension of the game (Konzack, 2002).

The first reason is still a general concern. However, due to the expanding success of what is often referred to as 'indie games', and the popularity of open-source software, more source codes are published and thus available to the researcher (Lipkin, 2012). Therefore, the method of source code analysis seems especially relevant for the study of games authored by individuals or small teams, and where the auteur-style mode of production makes it more meaningful to search for an authorial intent. Oftentimes, these games are much less complex in terms of source code than e.g. AAA games, which makes them more easily accessible for analysis, as the complexity and length of the source code influences its readability and the time it takes for the scholar to get an overview and understanding of the underlying structures. As such, the complexity and length of the source code may directly influence the possibility and likelihood of using the code in a formal analysis, as the source code of some games will be more readily accessible than others. The second reason is one of the main motivations for this paper; I wish to illustrate that it is possible to conduct a hermeneutic reading of the source code, which is studied in relation to the executed game, without having to dive deep into algorithms or understand complicated syntax. Quite the contrary, it should be possible for most game scholars who have just a little experience of reading or writing code to gain valuable insights from the analysis of the source code from the game in question, as a significant part of the analysis in this paper focuses on the naming of variables and simple representations of rules actualised in the executed game. I wish to illustrate exactly how this method is useful for game analysis by studying *Passage* (Rohrer, 2007), which has already been a target for other researchers exploring the link between computer science and game

studies (Robinson et al., 2015). The game has also been studied outside of computer science, e.g. as experiential metaphor (Harrer, 2013), as an art game, specifically in relation to Jason Rohrer's author statement (Parker, 2012), and in relation to the proceduralist line of thought (Treanor & Mateas, 2013). Because of the great interest from researchers, with various interpretations of the meaning or message of the game, it serves as the perfect case for illustrating how source code analysis can prove useful in game studies. I will argue that such a reading can be used to study metaphors and symbolism in a very different way than what only the executed game allows for. When combining the formal analysis of the executed game with the analysis of the source code, I believe that we can find stronger support for a traditional hermeneutic interpretation of metaphor and symbolism. As such, much of what is to be perceived by the player in the executed game is connected to elements in the code, which pose arguments for or against common interpretations. As such, this paper emphasises how games can be understood as second-order design, where the designed object is the code itself which, when actualised, produces a text in the form of the executed game. To this day, only few game scholars, including Mateas (2003), Wardrip-Fruin (2009), and Montfort and Bogost (2009), have used source code analysis as a method for studying games, and they have all approached the source code in various ways. One thing existing applications have in common is their focus on traditional authorial intent, which I will argue does not have to be an inherent part of source code analysis in the context of game studies. For the case of *Passage* this also means that the creator statement from Jason Rohrer (2007) should be unnecessary for a formal reading of the game, and I will illustrate how much of the information provided in this paratext is easily found in the source code. In the next chapter, I will outline the critique of authorial intent, in order to present how other scholars' readings of the game are all based on traditional notions of intentionality and authorship where explicit author statements are used to explore and understand the game object itself.

AUTHORIAL INTENT AND THE INTENTIONAL FALLACY

The notion of authorial intent is originally a part of literary theory and aesthetics. Its relevance has been, and is still, widely discussed in relation to literature as well as games. One of the most influential critiques of the increased focus on the author comes from Wimsatt and Beardsley (1946), who argue that "[i]f the poet succeeded in doing it [communicating the authorial intent], then the poem itself shows what he was trying to do. And if the poet did not succeed, then the poem is not adequate evidence, and the critic must go outside the poem—for evidence of an intention that did not become effective in the poem" (Wimsatt & Beardsley, 1946, p. 1-2). This argument is directly translatable to games; if the game manages to communicate what the author intended, there should be no need for studying this intention. If the game fails in doing so, the researcher must go outside the game, as done by Mateas (2003) in his study of *Pac-Man* (Iwatani, 1980), who bases his investigation on interviews with *Pac-Man*'s creator, Iwatani, to find evidence of this intention. But the discussion of authorial intent is complicated further when considered in relation to games, as it can be difficult and sometimes impossible to define the author on a production team of several hundred people. Who is to encode the game object with meaning, and who can then serve as "evidence" when investigating a potentially failed implementation of this meaning? Is it the lead designer, who is responsible for the overall vision of the game, the level designer who designed the specific level of the game, the artist who did the artwork for this level of the game, or the programmers who implemented parts of the algorithms behind the specific level of the game? Scholars who study the intention of the 'author' tend to focus on games developed by a single person (Montfort & Bogost, 2008; Robinson et al., 2015; Wardrip-Fruin, 2009), which is most likely because of the difficulties involved in identifying an author in AAA productions. If the author can't be defined, one cannot talk about the traditional notion of authorial intent, as known from literary theory.

In The Death of the Author, Barthes (1977) writes that "[…] it is language which speaks, not the author; to write is, through a prerequisite impersonality, […] to reach that point where only language acts, 'performs', and not 'me'" (Barthes, 1977, p. 143). The same can be argued for games where it is the game itself, including its hardware and software, interface, controller, and rule system, which speaks, acts, and performs, and not the 'game artist' or the 'author'. This perspective suggests the study of games, as Foucault identifies, 'works' (Foucault, 1969, p. 207). When studying a text as a 'work', Foucault argues that "the task of criticism is not to bring out the work's relationships with the author, nor to reconstruct through the text a thought or experience, but rather to analyze the work through its structure, its architecture, its intrinsic form, and the play of its internal relationships" (Foucault, 1969, p. 207).

Surprisingly enough, this seems to immediately align with the proceduralist school of thought, which is followed by scholars studying the source code of games, such as Mateas (2003), Wardrip-Fruin (2009), and Robinson et al. (2015); it is the game object itself, as a 'work', which speaks, and not its relationship to the author. However, Bogost argues that "[p]ersuasion is related to the player's ability to see and understand the simulation author's implicit or explicit claims about the logic of the situation represented" (Bogost, 2007, p. 333). In this, Bogost directly argues that the author has a relevant role, and that the persuasive power of a game is directly related to how the player understands the author in relation to the work. This is a complete contradiction to Foucault's critique of authorial intent as well as his approach to texts as 'works'. It therefore also makes sense that many of the scholars who build their arguments on the concept of procedural rhetoric tend to over-emphasise the role of the author in what may otherwise be understood as formal analyses, like Wardrip-Fruin's (2009) study of *Façade* (2005), Mateas' (2003) analysis of *Pac-Man* (1980), and Robinson et al.'s (2015) reading of *Passage*. They all focus on the author and his intent, in relation to how the game object is designed to convey meanings, and thus fall under, what Wimsatt and Beardsley call, 'the intentional fallacy' (Wimsatt &

Beardsley, 1946). This sort of focus on intent is old-fashioned and was abandoned in most fields quite a while ago. One can therefore wonder why it is still the dominant mode within source code analysis in game studies.

About *PASSAGE*

Passage is game designer Jason Rohrer's third game, which was developed for Kokoromi's curated GAMMA 256 event (Rohrer, 2007). The side-scrolling minimalistic game has an atypical narrow field of vision, where the player can only see limited horizontal "rows" of the map, although the world continues beyond what is immediately visible. The game consists of one level only which is procedurally generated, and which the player can decide to traverse either with an individual avatar or with a duo consisting of the avatar and a spouse. The spouse is activated when the player collides with the graphical representation of the spouse on the map. After the spouse has been picked up, narrow paths can no longer be traversed, and thus fewer points can be collected through collision with treasure chests (some of which contain valuables that are translated to points, while others contain something resembling flies or dirt). Points are earned by colliding with treasure chests and simply by progressing through the level, and while it is harder to gain points from chests when having picked up the spouse, this mode of play grants an additional point bonus, in the documentation referred to as the *explorer factor*.

Figure 1: Screenshot of Passage (2007)

When the game is started, the avatar is located to the far left of the screen, with a blurred part to the far right of the screen, which is revealed through play to be the remaining level. As the player progresses, the

avatar moves further to the right of the screen, all the while its graphical representation (and that of the spouse, if she is picked up) seems to age. Eventually the avatar will move slower, and if the spouse has been picked up she will transform into a gravestone. The avatar too will transform into a gravestone and the game will end after exactly 5 minutes, regardless of the space traversed during play.

APPROACHES TO *PASSAGE*

As previously stated, several scholars have studied *Passage*, all sharing the interest of how the game conveys meaning. Parker (2012) argues that *Passage* is the first prominent 'artgame', for the definition of which he emphasises identifiable author figures, as well as a specific 'message' which the player is to discover (ibid, p. 42). Although Parker argues that such characteristics are not mandatory for a game to qualify as an artgame, it is exactly the author and the intended message of *Passage* which he studies in his article. He identifies what I will later uncover as the spouse, the female-looking character that follows the player after colliding with her on the map, as a companion. This is linked to what Parker identifies as the autobiographical character of Rohrer's games – Rohrer articulates in his creator statement that the game is based on his thoughts about life and death. However, Parker also notes the irony of the many readings of *Passage*, as Rohrer states that there is no right or wrong way of interpreting the procedurally generated game. This is somewhat contradicted by Rohrer's presentation of his own intentions, that has, after being published in the creator statement, guided most readings of the game.

Another scholar who has engaged with *Passage* is Harrer (2013), who in an initial formal introduction of the game starts interpreting the symbolism of the executed game: "[r]unning into her triggers an animated heart that represents their falling in love" (Harrer, p. 616), she argues, acknowledging the existence of the creator statement, yet aiming her analysis at the symbolism and metaphors of the game, rather

than what Rohrer intended (or, at least, argues in his creator statement that he intended with the game). When Harrer discusses the game as an experiential metaphor, however, she leans on the creator statement, using Rohrer's quotes to enforce her point that the death of, what she (correlating with Rohrer's creator statement) terms the spouse comes, as a shock to the player.

At the DiGRA conference 2015, William Robinson, Michael Mateas, and Dylan Lederle-Ensign presented a reading of *Passage*. They argued for what they term a procedurally literate inspection (Robinson et al., 2015) in which processes are read as metaphors. Their approach is grounded in Bogost's theory of procedural rhetoric (Bogost, 2007), and follows the notion that meaning can be found in the game object itself. An author encodes this meaning and hence it should be studied in relation to the author's intentions. Therefore their study is based on Rohrer's creator statement, which has its limitations; Rohrer describes his intentions with the game, but does not go into details about the specific code and how it reflects the meanings that he wished to encode in the game. Uncovering the meanings inscribed by Rohrer in the source code becomes the mission for Robinson et al., who end up having to draw potential conclusions, as they do not have the necessary knowledge from the author to validate any of the results of their analysis, e.g. the goals of the game metaphorically relating to everyday trade-offs (Robinson et al., 2015, p. 3). It is exactly this problem Wimsatt and Beardsley (1946) refer to when they argue that the poem, or in this case the game, is not sufficient evidence in supporting any argument about the authorial intent. Although Robinson et al.'s reading is indeed interesting, it does not explore the full potential of the source code in relation to their metaphorical reading of the game. Instead, they conduct a metaphorical reading of the code itself, not directly related to the executed game, and one can therefore question the study's relevance for the study of the game artefact, often conceived of as the executed game itself.

All the readings briefly introduced above build, in one way or another, on Rohrer's creator statement, acknowledging his authorial intent as core

to an analysis of *Passage*. Parker's and Harrer's studies are focused only on the executed game, that is, that which is played and perceived by the player, and what is usually simply referred to as 'the game', while Robinson et al.'s study deals with the source code of the game. The aim is now to illustrate that similar interpretations can be made from a reading of the source code, rather than based on the creator statement. As such, the goal is to eliminate the notion of traditional authorial intent in the analysis, and thereby not studying Rohrer's creator statement as a paratext.

ANALYSIS

I will demonstrate, with *Passage* as an example, that we can analyse the source code in a meaningful way, and that the findings can be logically connected to interpretations of the executed game. When analysing the code we do not have to get involved with the intentional fallacy per se. If we acknowledge the executed game as second-order design, we must also accept the relevance of the source code as the mother-text, that which is actually designed, and thus it becomes meaningful for the study of the executed game. However, in this case one may understand the source code as an author itself, or as a type of creator statement, in which my initial distance to the notion of authorial intent becomes problematic. I will return to this problem in the discussion, but for now, I will conceive of the source code as a text between author and game, which may help us understand better the executed game itself.

Code offers many aspects and dimensions that can be analysed. For the sake of clarity and conciseness, I will focus the discussion in this paper on naming in the context of processes as well as processes, which constitute rules that are not necessarily clearly articulated in the executed game. Naming refers to the names of variables and methods in the code. These are given by the programmer, but most programming languages allow for meaningless combinations of letters to form the names (Deissenboeck & Pizka, 2006). Many programmers and scholars argue

for a set of unified rules that dictates exactly how one should name variables and methods, as this has a direct influence on the readability and in turn the overall program comprehension (ibid). Although there are no fixed rules for naming, there are some norms: there should be consistency in whatever method the programmer uses for naming, the names should be concise, and method names should, if possible, be formed following a verb-noun or verb-noun-noun structure (ibid). Naming can be seen as a part of the code aesthetics, and can therefore be studied in relation to program comprehension, however in the following analysis, I will search to make sense of method and variable names as parts of a formal analysis of video games. My argument is that much meaning can be interpreted from the code itself, and that much of this meaning may not be immediately evident in the executed game. As such, some of the things that are usually interpreted on an abstract and symbolic level in the traditional game analysis will be hardcoded into the game's source code through naming. I believe that this allows us to arrive at informed interpretations without turning to creator/author statements and other secondary sources in which the programmers and/or creators express their intent and meaning of the game. It should be noted, though, that it is of course possible for the designer to name variables freely and thus that variable names, like any other sign, must be interpreted in the context of the executed game. If variable names reflect certain ideologies, e.g. socialism, without these being in any way visible in the executed game, it does not mean that the game can or should be understood as a socialist game. Rather, the scholar must explore the relation between the variable names and the executed game, and she may even find that there is no apparent connection between the two. Yet, as the source code is a part of the formal game object, it can (and possibly should) be understood in relation to the game as a work – something I will return to in the discussion. This also means that two different source codes, which produce the same executed game, must be considered as two different works.

As with any other game example, large parts of *Passage's* code are not relevant to this investigation. The few lines that are useful for the

analysis have been found by playing through the game, interpreting symbolism and metaphors, and returning to the code to see if any of this is spelled out in the code. Moreover, as I will present below, I attempted to compare my findings to Rohrer's creator statement to see how it relates to my findings. This is not to argue against my previous statement that authorial intent in some cases is unconstructive – rather it is to prove that some of the interpreted meanings of a game can be found in the code itself and do not depend on any communication with the "author" or "creator". It should be noted that the source code to *Passage* is somewhat difficult to access[1]. The game has a built-in script, which links and compiles various files together, rather than a traditional set-up with one folder containing the various classes. I have therefore only investigated Game.cpp, containing 1300 lines of game logic code, which makes calls to various other scripts, for example, graphics, sound, etc.

```
945
946         if( isPlayerDead() ) {
947             // stop moving
948             moveDelta = 0;
949             }
950
```

Figure 2 – Excerpt from Passage (2007), line 946-949 in Game.cpp

```
1166
1167         if( age >= 0.85 ) {
1168             dieSpouse();
1169             }
1170         if( age >= 0.95 ) {
1171             diePlayer();
1172             }
1173
```

Figure 3 – Excerpt from Passage (2007), line 1167-1172 in Game.cpp

1. I owe thanks to Dylan Lederle-Ensign, who provided valuable hints for navigating Passage's file structure.

When the game is played, the player will find that the avatar changes graphically, indicating aging. At some point, the avatar will turn from being a graphical representation of a human to being a graphical representation of a gravestone, as pointed out by Harrer (2013). This kind of signification is unambiguous to many, yet the example above illustrates how it is also evident in the source code—what we as players interpret from the symbolism of the gravestone as death of the avatar is clearly written in the script as death of the player, which will make the avatar stop moving. Figure 3 illustrates how both spouse and player (avatar) will die when reaching a specific age, calculated based on the time played – that is, even if the player does not move the avatar around, it will age, and eventually die when its age value reaches 0.95, an arbitrary number which does not necessarily refer to the actual age of 95 years. The two examples above serve as examples of how code reading in the analysis can support arguments of interpretations and contribute with new meanings. It also illustrates a strong relationship between player and avatar, which could be interpreted as if there is no character in the game, but only an avatar, which is meant as a graphical representation of the player and is not articulated as a manifestation of Rohrer himself. The example above also reveals that the gameworld does not end as such, but that the avatar turns into a gravestone and stops moving once dead. This is articulated in Rohrer's (2007) creator statement, where he writes that "[…] even if you spent your entire lifetime exploring, you'd never have a chance to see everything that there is to see" (ibid, paragraph 8). However, the code makes that visible to us (also articulated in the map generation script), hence we do not need the creator statement to figure that out. The meaning of being in an infinite world, where you will not have enough time to explore everything is to be interpreted by the scholar, if she wishes to do so, but the potential endlessness of the graphics, and the fact that not all can be seen, is hard-coded in the source code itself, and so is the length of the avatar's life, emphasising the player's lack of autonomy, both in the game as well as metaphorically in life.

```
950
951         if( knowSpouse && isSpouseDead() ) {
952
953             // player moves slower
954             // toggle motion on this frame
955             movingThisFrame = ( frameCount % 2 == 0 );
956             }
```

Figure 4 – Excerpt from Passage (2007), line 951-956 in Game.cpp

This second example illustrates how the 8×8-pixel human, which follows the player once the player, collides with its graphical representation, triggering a heart animation (see fig. 5 below), is presented as a spouse. This is something, which the executed game never articulates, but which Rohrer explains in his creator statement. Parker (2012) interprets the spouse as a companion, whereas Harrer (2013), possibly influenced by the creator statement, uses the same term as Rohrer, namely spouse. It is possible for the player to interpret the human companion as a spouse, a wife, a friend, a companion, or whatever she wishes, but it is stated in the code that this being is the spouse. This means that the scholar should, when studying the source code as a part of the formal analysis, consider the meaning of the term ‘spouse’ in relation to the graphical representation, to explore how the variable name can inform the interpretation in question. Moreover, the code example shows how the death of the spouse will result in slower movement of the avatar. This decrease in speed is only activated if the player picks up the spouse and she dies, and therefore the player will not be slowed down if she decides not to pick up the spouse. This is something that Rohrer explains in his creator statement, but which is articulated as a process in the source code, and thus an example of how rules, which may not be immediately clear in the executed game, can be better understood by looking into the code.

```
1202
1203        if( ! haveMetSpouse() &&
1204            ! isSpouseDead() &&
1205            distanceFromSpouse < 10 ) {
1206
1207            meetSpouse();
1208
1209            knowSpouse = true;
1210
1211            startHeartAnimation(
1212                (int)( ( spouseX - playerX ) / 2 + playerX ),
1213                (int)( ( spouseY - playerY ) / 2 + playerY ) - 2 );
1214            }
1215
```

Figure 5 – Excerpt from Passage (2007), line 1203-1214 in Game.cpp

The code excerpt in fig. 5 shows that the player can only ever have one spouse during a play-through, as the heart animation, which is triggered when colliding with the spouse and picking her up, is only generated if the player has not yet met the spouse, the spouse is not dead, and the spouse is not next to the player (that is, the spouse is not currently active). That also means that the game only ever generates *one* instance of the spouse per play-through. This is never commented upon in the creator statement, but can indeed inform the formal analysis of the game, as monogamous, heteronormative standards are hardcoded into the game. Depending on the reading, one might even argue that this conveys that there is only *one* right person for you in the world. This specific example illustrates how an analysis of the source code can not only exclude the use of creator statements and interviews, but also contribute to the analysis of games in new and meaningful ways.

```
1240
1241        if( haveMetSpouse() ) {
1242            // show explore score contribution in jumps
1243            exploreScore =
1244                ( exploreScore / spouseExploreFactor )
1245                * spouseExploreFactor;
1246            // note:
1247            // this can cause our score to go down (to the previous
1248            // jump) as we transition from not having a spouse to
1249            // having one.
1250            // we fix this below with maxExploreScore
1251            }
```

Figure 6 – Excerpt from Passage (2007), line 1241-1251 in Game.cpp

```
1003
1004        if( getKeyDown( SDLK_LEFT ) || getJoyPushed( SDL_HAT_LEFT ) ) {
1005            char notBlocked =
1006                !isBlocked( (int)( playerX - moveDelta ), (int)playerY );
1007
1008            // spouse and character move, and are blocked, together
1009            if( haveMetSpouse() &&
1010                isBlocked( spouseX - moveDelta, spouseY ) ) {
1011                notBlocked = false;
1012                }
1013
```

Figure 7 – Excerpt from Passage (2007), line 1004-1012 in Game.cpp

The fifth example (fig. 6) from the source code of *Passage* shows how points are calculated differently if the player picks up the spouse. A certain “spouse explorer factor” (which is previously given the value of 2 in the code) determines exactly how the overall explorer points are calculated. The score will always be twice as high with the spouse and her explorer factor than without her. Along with the way the spouse blocks your possibilities of navigation on the map, as illustrated in the code in fig. 7, this creates a situation where there are points to be gained both with and without the company of the spouse. Rohrer too accounts for this in the creator statement, but rather than commenting on the meaning of this way of collecting explorer points, he presents the scoring-system as a part of the formal rule-system of the game. Yet again, this proves that the creator statement is not needed to make many of these conclusions.

DISCUSSION

As can be seen in the analysis, I have found different ways in which the source code is useful for a formal analysis of the game. The most crucial way in which source code analysis facilitates more in-depth understanding is how it can help the scholar uncover dimensions of the game object, which are not necessarily visible in the executed game. This is illustrated in the examples presented above, and as noted in the analysis, these cases fall under two categories: 1) cases in which the

specific variable or method name reveals information about the game object and its potential representations on the interface, and 2) examples that show how specific implementations of procedures establish rules in the game that cannot necessarily be seen in the executed game. The first category can support interpretations of the executed game in the sense that it contributes to an understanding of metaphors and symbols, as the naming of the variable or method may help the scholar to unravel a potential reading of the game. A good example is the name of the spouse; not only does this naming clarify our understanding of the companion-like pick-up, it also guides further readings of the code where the companion-as-spouse contributes to deeper readings, such as the idea that the only-once-generated and picked-up spouse may reflect a certain sense of heteronormativity. This method can be useful in analyses where the symbolism is especially ambiguous and hard to interpret. Moreover, it enforces a focus on games as second-order design – that which is designed by an author is not the executed game but the source code. Hence, it is fruitful to study whether there is a correlation between interpreted messages and meaning in the executed game as well as in the source code. This, I believe, is especially true if we wish to isolate the analysis from the author's intentions, goals, "points", etc., as it allows us to answer the question of whether any or all of the two levels of the game that can be studied formally appear to communicate a specific message or can be attributed a specific meaning. Yet, as previously argued, the source code itself can be understood as an author, which influences how we dare interpreting the executed game. This creates a new paradox of an intentional fallacy, where the scholar must explore the relationship between code and executed game to assess whether variable names are at all worthwhile or reliable to study in the formal analysis.

In the second category, the source code is used to get a perfect understanding of the system structure and it draws some resemblance to the fan practice of theorycrafting. However, whereas variable names can be understood as a type of pseudo-representation, the rules constituted by processes do not signify anything in themselves. Only when understood in relation to the executed, playable game they gain meanings, as they

constitute the borders of play. Because these rules are not necessarily visible in the executed game, they raise the question of whether they, and the whole of the source code, can truly be understood as a part of the 'work' of the game, of whether it is simply another form of paratext that can be explored similarly to author/creator statements.

The Foucauldian meaning of the 'work' emphasises an analysis of the structure, architecture, intrinsic form, and internal relationships of the work itself, rather than of its relationship to the author (Foucault, 1969). The analyses found in this paper do indeed study the structure of the game, which can be perceived more structurally and numerically in the code than in the executed program. Moreover, the source code facilitates an exploration of the internal architecture of the game, and not just that which is represented when executed and played. All of this relates to Foucault's definitions of what constitutes a 'work'. However, the source code is not the executed game object. We have to accept that we are, when working with source code analysis as a part of a formal game analysis, working with different texts. The relationship between these texts can be understood in various ways; the source code can be seen as the mother-text and the executed game as a text designed by its 'parent-text'. The executed game can be seen as the main text, whereas the source code can take on the role as another text related to the main text, e.g. as a hypotext or paratext. The source code itself can be understood as several texts, as the source code consists of respectively, code and comments, which may be understood as separate texts, which are ontologically different. With the code as a hypotext, the analysis can explore the executed game as a subsequent text to the source code. This would indicate that the executed game is not authored as a part of the source code, but that their relation is influenced by, among other things, the compilation and execution of the program. The many complicated ways in which we can understand the relations between code, executed program, and player have been studied in depth by Aarseth (1997). He suggests the concept of cybertext, which prioritises the influence of the medium on the dynamics of scriptons (defined as strings as they appear to the reader) (ibid, p. 62-63). Other scholars have searched to

make sense of the source code's place in the text/paratext relationship (Desrochers, 2014). In relation to the work conducted for this paper I will argue that all approaches seem somehow productive, yet they all pose a question of authorship, and create a paradox of how to dismiss the notion of the intentional fallacy. If the source code is seen as a paratext, it does not solve any problems of authorial intentions; rather it facilitates further discussion of authorial intent understood as authoring of the code or the authoring of the executed game, and whether we can talk about one or several individual authors, or one heuristic author. As a paratext, the code is only as relevant for the formal analysis of the game as an author statement or wiki page, because of its distanced relationship to the text or executed game in question.

Neither hypotext, cybertext, nor paratext may be the right ways to make sense of the relationship between the game and its code, and hence what status the source code should have in the formal analysis. However, I believe that the discussion of source code as a text illustrates that more research is still needed to point out exactly how source code analysis can be situated as a method in game studies. I also believe that I have demonstrated how such readings of the code can inform the formal analysis. The next step must be to unravel how a full integration of the method for a comprehensive game analysis contributes to the academic work. This should be followed not as much by a specific study but rather by a discussion of the relations between code, game, and everything in between.

Compared to other readings of *Passage*, I believe that the analysis of the source code is indeed a valuable way of studying several dimensions of the game object. I have illustrated that it allows us to consider elements that contribute to an understanding of 'message' or 'meaning', without engaging with creator statements or other traditional authorial documents. As such, it is possible, at least to some degree, to focus on the game object itself, rather than intentions of the authors, which allows for a more formal analysis. The source code analysis may not give us more information than the author statement would, yet it will be a

different type of information, which can exclude the use of second-hand references, such as the author statement. However, as previously pointed out in the discussion, the source code consists of comments as well as code, and the author can thus still express intentions through comments and variable naming. It is therefore necessary to further study exactly how we can understand the source code as a text, or several texts, in relation to the executed game, and in turn how the source code can be understood as different from author statements.

CONCLUSION

This paper has illustrated how source code analysis can serve as a tool for a formal game analysis. By analysing the source code of *Passage*, I have identified two ways by which such an analysis proves useful. First, the variable or method names can reveal information about the game object and its representations in the executed game, which supports an analysis of symbolism and metaphors. Second, the code can contain processes, which are not necessarily visible in the executed game. Exploring these processes can be likened to practices of theorycrafting, as it helps us better understand the system structure of the code and hence the game. Both ways of exploring the source code can be applied in the traditional formal game analysis and contributes with new meanings.

I believe that the study proves that a reading of the source code is useful for a comprehensive analysis of a game, and that it may help avoid the intentional fallacy of prioritising author statements over close analysis of the text. However, we must methodologically understand the approach in relation to the game object, that is, explore exactly how we can understand the code in relation to the executed game. Moreover, in order to justify how source code analysis can avoid the intentional fallacy, we must study in depth whether game designers and programmers can be understood as heuristic authors, whether we have to accept the author as an individual, at least for cases where the game is made by one person,

or if there is a third, advantageous way of conceptualising the author(s) of video games. Until then, source code analysis poses some ontological challenges to the understanding of games as 'works'.

BIBLIOGRAPHY

Aarseth, E. J. (1997): *Cybertext: perspectives on ergodic literature*. Baltimore: JHU Press.

Barthes, R. (1977): *Image, Music, Text.* London: Fontana Press.

Bogost, I. (2007): *Persuasive games: The expressive power of videogames.* Cambridge: MIT Press.

Buckles, M. A. (1985): *Interactive Fiction: The Computer Storygame Adventure.* Doctoral Dissertation, University of California, San Diego.

Crowther, W. & Woods, D. (1977): *Colossal Cave Adventure.* [PC/Unix]

Deissenboeck, F., & Pizka, M. (2006): Concise and consistent naming. In *Software Quality Journal*, 14(3), pp. 261-282.

Desrochers, N. (2014): *Examining Paratextual Theory and its Applications in Digital Culture*. Hershey: IGI Global.

Foucault, M. (1969): What is an author. In *Aesthetics, method, and epistemology* (Vol. 2), ed. by Faubion, J.D., 1998. New York: The New Press

Harrer, S. (2013). From Losing to Loss: Exploring the Expressive Capacities of Videogames Beyond Death as Failure. In *Culture Unbound: Journal of Current Cultural Research*, 5(4), pp. 607-620.

Iwatani (1980). *Pac-Man*. Tōru Iwatani/Namco.

Konzack, L. (2002): Computer Game Criticism: A Method for Computer Game Analysis. In *Proceedings of Computer Games and Digital Cultures Conference*.

Lipkin, N. (2012): Examining Indie's Independence: The Meaning of" Indie" Games, the Politics of Production, and Mainstream Co-optation. In *Loading… The Journal of the Canadian Game Studies Association*, 7(11).

Mateas, M. (2003): Expressive AI: Games and Artificial Intelligence. In *Proceedings of Level Up: DiGRA 2003.*

Montfort, N., & Bogost, I. (2009): *Racing the beam: The Atari video computer system.* Cambridge: MIT Press.

Parker, F. (2012). An Art World for Artgames. In *Loading… The Journal of the Canadian Game Studies Association*, 7(11).

Procedural Arts (2005). *Façade.* [PC] accessed 09.12.16 at http://www.interactivestory.net/

Robinson, W., Lederle-Ensign, D., & Mateas, M. (2015): Procedural Deformation and the Close Playing/Reading of Code: An Analysis of Jason Rohrer's Code in Passage. Abstract from *DiGRA 2015: Diversity of Play*.

Rohrer, J. (2007): *Passage*. [PC] accessed 20.01.16 at http://hcsoftware.sourceforge.net/passage/

Rohrer, J. (2007): *What I was trying to do with Passage.* Web-post, published 2007, Potsdam, NY, accessed 03.12.15 at http://hcsoftware.sourceforge.net/passage/statement.html

Treanor, M., & Mateas, M. (2013). An Account of Proceduralist Meaning. In *Proceedings of the 6th International Conference of the Digital Research Association*.

Wardrip-Fruin, N. (2009): *Expressive Processing: Digital fictions, computer games, and software studies.* Cambridge: MIT press.

Wimsatt, W. K., & Beardsley, M. C. (1946): The Intentional Fallacy. In *The Sewanee Review* 54 (3), pp. 468-488.