Programmable Fiction

An experiment in developing an alternative authoring tool for interactive fiction

Kevin McGee
Department of Computer Science
Linköping University
581 83 Linköping, Sweden
kevmc@ida.liu.se

Tord Svensson
Department of Computer Science
Linköping University
581 83 Linköping, Sweden
torsv@ida.liu.se

Keywords

Interactive Fiction, Interactive Narrative, Authoring Tools, Programming Language Design, End-User Programming, Hypertext

ABSTRACT

Successful traditional narrative is often described as a balance between those aspects that are familiar and those that are new; between the degree to which those aspects are resonant with reader *dispositions* towards form or content – and the degree to which they *indispose* readers, unsettling them in various ways with engaging novelty. Perhaps the development of similarly engaging, novel works of *interactive fiction* could benefit from more expressive authoring/developing tools. This paper describes some work we have done to explore this in terms of a system for developing works of interactive fiction that give readers/players of such works the ability to *write programs* as part of their interaction with the stories/games.

1. INTRODUCTION

What can we do if we want to create interesting works of *interactive fiction* (IF) – works that are something other than "playing a game with a back-story" or "providing a joystick for a narrative experience"? Surveying the majority of IF works, one is struck by their similarity to each other; by and large, the genre as a whole does not yet seem to display traditional literature's diversity – nor does it seem to have "come into its own", successfully fusing and transcending its twin origins.

One approach to innovation is familiar from traditional story-telling: "make a theory." There is a great deal of work in this vein, to identify aspects of games, narrative, and technology, and propose theoretical models of good (new) form [7, 8]; there is also some fascinating applied IF research, with theories realized in a variety of interesting systems [3, 10, 13, 11, 12, 6, 4]. However, although there is a growing body of theoretical work about IF, there does not yet seem to be any fully satisfactory means of facilitating the creation of innovative IF works. Indeed, the history of story-telling suggests that narrative innovators often feel they can do quite well without a *theory*, thank you very much; practitioners often innovate in the *praxis* of creating stories (whether or not such innovative praxis is based on "having an implicit theory" is another story, so to speak).

So, although there has been a growth in recent years of IF *theory*, it may be also be fruitful to study and develop the expressiveness of the *tools* that are available to IF authors – tools that may allow them to experiment with *creating* alternatives. To be sure, there are excellent authoring tools for IF, and there is even innovative research in this area [5]; but by and large, IF tools are designed to support the creation of works that involve a reader/player-as-object interacting with and manipulating a world of objects. One could say that this model of *objects* that can change *state* almost entirely dominates the genre – not only in terms of the works created, but also in terms of the tools for authoring such works. The majority of languages for implementing IF works tend to be strongly *imperative* and *object-oriented*. This may be appropriate for IF that tends to be linear[1], includes various kinds of objects (rooms, containers, agents, and the like), and is *event-based*. But it does tend to impose constraints for authors who would like to develop alternative forms. Indeed, IF newsgroups regularly contain discussions about interesting design-changes that developers would like to explore in the creation of specific works – and complaints that existing IF languages are not flexible enough to allow such experiments without major effort.

So, one possible contributing factor is the similarity of *paradigm* among the different authoring-languages for IF – a similarity which may contribute to a similarity in the works created with those otherwise different languages. Of course there are innovative examples of IF created with such languages. However, it also seems possible that different programming paradigms can help inspire different, innovative IF *forms*. Said another way, different language paradigms make it *practical* (i.e., realistic) for programmers to build different classes of programs – and if one particular programming language makes it easier than another to, say, use recursion or higher-order functions, we might expect such techniques to inform the kinds of the stories/games in different ways.

It might therefore be worth developing IF authoring-tools based on a variety of programming paradigms. Such tools could contribute to the discovery and development of innovative game *and* narrative forms – forms that are very different from existing examples of games and narratives. And this, in turn, could hopefully contribute to the devel-

[1]By "linear" we mean that the IF work is structured as a *sequence* for players – puzzles to solve or "experiences" sequenced according to some model of a satisfying narrative arc – whatever other non-linear options players may have.

Presented at Hyperstructure 2005: Workshop on Narrative, Musical, Cinematic and Gaming Hyperstructure – held in conjunction with Hypertext 2005, September 2005, Salzburg, Austria.

opment of IF as a genre consisting of substantially more than "moving player-objects through an appropriately arranged sequence of experiences – from place to place within a larger, well-structured place (world)."

Such a research interest may seem slightly ironic in the context of *dispositio*, rooted as it is in the concept of *place*. But this idea of "placing" story/structure elements in some arrangement is only one aspect of *dispositio*. Another aspect, especially as it relates to the creation and experience of works of art, involves *dis*-placement. That is, works of art are often successful precisely because they resonate with our *dispositions* while simultaneously *indisposing* us in various ways.

This idea has, of course, been the basis of numerous insights in narrative theory. For example, twentieth century practitioners and theorists such as Bertolt Brecht and Roland Barthes have made much of the ways in which innovative narrative techniques can make narratives "alien", thus facilitating reader/ audience awareness of stories-as-stories. And in this awareness, they claim, is a deeper appreciation of narrative: story *structure* can make us aware of narrative conventions and cognitive expectations – and by subverting them in various ways, engage us as readers. Similar claims are also made for successful games: that they are a felicitous blend of rules and chance – of control and uncertainty.

To a certain extent it would be possible for IF works to attain this through techniques of "structured randomness" (in the spirit of, say, John Cage or "happenings"). However, an alternative is the development of appropriate *authorial* techniques for unsettling or surprising the audience – without alienating the audience to the point where the works are completely unrecognizable or meaningless. Such techniques could be relevant to a wide variety of IF concerns, including those that emphasizes innovation on the part of either the author/designer, the reader/player, or the "encounter" between author, reader, and work.

In order to begin exploring the potential of expressive tools for creating innovative IF works, we have been working on the implementation of a small IF authoring-language for creating a particular class of IF-works. In particular, we have been interested to see "what kinds of interactive stories we can *think* with a different *kind* of authoring tool." To do this, we implemented a small authoring environment with Scheme, a predominantly *functional* programming language that is particularly good for designing other programming languages. Thus, one of the things "we have thought" has been been the possibility of creating IF works that encourage readers/players to *write programs* as part of reading/playing. This leads to the second point about the relationship between IF works, authors, readers, and encounters: the potential of such IF works raises a number of interesting suggestions about the possibilities of readers/players to actively participate in the creation of the works they read/play.

Of course, in recent years, some work has been done to make it possible for game-developers to more easily extend the underlying game-engines [2] and script the behaviors of "non-player characters" (NPCs) [9]; it is also becoming more common for certain classes of games to support player-scripting of player-controlled agents or groups (e.g., squads, divisions, armies). However, there does not seem to be much work on developing IF works that support the use of programming by players/readers of such works. Thus, we have explored this as an interesting feature for an IF authoring tool to support.

The remainder of this paper briefly describes our method, the implemented system, an illustrative example of playing the implemented game, some limitations of the current implementation, and some insights about IF-language design based on the work to date. The paper concludes with a broader discussion of some ways that programming language paradigms can inform the creation of interactive narrative.

2. RESEARCH METHOD

Our method may be summarized as follows. Choose an implementation language, build a version of the standard IF *reference game* ("Cloak of Darkness") in our implementation language, and then experiment with different transformations to the game that derive from the possibility of giving players the capability of writing programs as part of playing the game – that is, we implemented different transformations to the game-world and game-play that were suggested by concepts, mechanisms, and techniques from the implementation language.

Since Scheme is not commonly used for game-development, it is worth saying a few words about this choice. Of course it is possible (and common) to implement languages in languages other than Scheme. So, in one sense, the choice of Scheme merely provides a flexible and powerful prototyping language that allowed us to experiment with a variety of different implementations of the game-world and possibilities for players. On the other hand, the choice of Scheme did also provide *inspiration* and ideas for the design of the world and player-language that might not have been the case if we had chosen, say, Java or C^{++}, as the implementation language. In particular, the choice of Scheme helped us to think about the design of the game-world in terms of *meta-programming*. That is, to think about the game-world as a *program* – and the player's role in terms of *the creation and execution of different classes of programs*. These different classes of programs might include such things as interpreters, shells ("command prompts"), operating systems, various kinds of self-referential and self-modifying meta-programs, and even *programming languages* themselves.

With this in mind, the choice of the actual *implementation* (or game *developer*) language also led to a consideration of different kinds of programming techniques and mechanisms afforded by that language; and this, as we will see, in turn informed the design of the actual game.

2.1 Reference Implementation

The IF community generally accepts a particular reference implementation as a *minimum* requirement for an IF language. The implementation is known as *Cloak of Darkness* and is briefly described below.

> "Cloak of Darkness" is not going to win prizes for its prose, imagination or subtlety. Or scope: it can be played to a successful conclusion in five or six moves, so it's not going to keep you guessing for long. There are just three rooms and three objects.

- The Foyer of the Opera House is where the game begins. This empty room has doors to the south and west, also an unusable exit to the north. There is nobody else around.
- The Bar lies south of the Foyer, and is initially unlit. Trying to do anything other than return northwards results in a warning message about disturbing things in the dark.

- On the wall of the Cloakroom, to the west of the Foyer, is fixed a small brass hook.
- Taking an inventory of possessions reveals that the player is wearing a black velvet cloak which, upon examination, is found to be light-absorbent. The player can drop the cloak on the floor of the Cloakroom or, better, put it on the hook.
- Returning to the Bar without the cloak reveals that the room is now lit. A message is scratched in the sawdust on the floor.
- The message reads either "You have won" or "You have lost", depending on how much it was disturbed by the player while the room was dark.
- The act of reading the message ends the game.

  Source: `http://www.firthworks.com/roger/cloak/`

Figure 1 illustrates some of the game-states, player-options, and player-commands in a brief example-game of *Cloak of Darkness*.

2.2 Limitations

For this project, we did not focus on the problem of designing either an appropriate *developer* or an appropriate *player* language. Each of these is an interesting problem in its own right, and one to pursue in the future. For now, the focus is on the *kinds* of world features and phenomena inspired by a language such as Scheme – and on the *kinds* of things that a player/programmer might be able to do with such a language.

Thus, we were not concerned with issue of developer-language design – or how to learn (or how easy it is to learn) the particular developer-language.[2] Similarly, there are a number of ways we could modify the design of the player-language to make it more familiar (few parenthesis, infix notation, etc.). But for this research, we assume that both developers and players of our example game are reasonably familiar with Scheme itself.

3. SYSTEM DESCRIPTION

For this system, we implemented *Scheme of Darkness* (*SoD*), a variation of *Cloak of Darkness*. Although the complete design and implementation of *SoD* involves a number of different game-play features and constraints, for this paper we will simply highlight the fact that it is a *multiplayer* game in which *players can create programs as part of playing the game*. In fact, players can treat most of the game elements – and the *game-engine itself* – as programs they can modify in real-time. Below we describe some of the main features of *SoD*, describe an example game in narrative form, and show an example of that brief game.

- In *SoD* two players can compete with each other to be the first player to a) remove cloak in Cloak Room, and then, b) read the winning message in the Bar. They can take various actions to achieve this goal – and may also take actions to prevent the opponent from winning first.

[2]Note, however, that we *have* created a suitable developer-abstraction for ourselves – one that we use to implement the end-user language.

```
Hurrying through the rainswept November night,
you're glad to see the bright lights of the Opera
house. It's surprising that there aren't more
people about but, hey, what do you expect in a
cheap demo game...?

FOYER OF THE OPERA HOUSE
You are standing in a spacious hall, splendidly
decorated in red and gold, with glittering
chandeliers overhead. The entrance from the street
is to the north, and there are doorways south and
west.

> west

CLOAKROOM
The walls of this small room were clearly once lined
with hooks, though now only one remains. The exit is
a door to the east.

> hang
(the velvet cloak on the small brass hook)
(first taking it off)

You take off the velvet cloak.
You put the velvet cloack on the small brass hook.

[Your score has just gone up by one point.]

> east

FOYER OF THE OPERA HOUSE

> south

FOYER BAR
The bar, much rougher than you'd have guessed after
the opulence of the foyer to the north, is
completely empty. There seems to be some sort of
message scrawled in the sawdust on the floor.

> read
(the scrawled message)
The message, neatly marked in the sawdust, reads ...

    *** YOU HAVE WON ***
```

Figure 1: *Cloak of Darkness*, sample game

- The initial world is largely the same, consisting of the Foyer, the Cloak Room, and the (darkened) Bar. When the game begins, both players are in the Foyer. Each player then tries to be the first to win the game.
- In addition to such traditional commands such as `move`, `look`, `drop`, and the like, it is also possible for players to create different kinds of programs. These programs can range from relatively simple *aliases* for existing commands, to simple scripts that transform multiple commands into a single (named) command, to more complex programs that create alternative behaviors for existing game-commands, to even more complex programs such as false game-prompts.

  In fact, players are able to programmatically change almost anything about the world: they can rebind game-commands to new commands, redefine existing commands, and modify the existing game-elements and behaviors.[3]

[3]*SoD* also differs from the original by the fact that it in-

- Each player has a separate window onto the shared world. Players do not see what each other type, but only the results of another player's actions. This means that players do not see whether/how the other player creates new programs. However, both players have *access* to any programs created by either player (i.e., by "guessing" or trying possible commands).

  With one exception, play is turn-based, with control switching automatically between players. The exception: when a player simply defines a new program it is not considered a full turn.

- *SoD* differs from *Cloak of Darkness* (and many games of that type) in that there is an assumption that players know the winning conditions before they start. It is not a mystery (or puzzle) in that sense; to the extent there is challenge in the game, it comes from anticipating and handling the inventiveness of the other player.

In order to illustrate several different kinds of program-creation in *Scheme of Darkness*, Figure 2 (at the end of this document) is a brief example-game between two players.

To situate the example, here is a brief narrative summary of the game. At first the two players do simple things: look at the inventory of things they carry, "sleep" (to see if anything interesting happens as "time passes"), and move from one place to the next. Then, Player 2 creates his own aliases for the *move* command and uses it. Then, by *guessing* the existence of the new alias, Player 1 moves by using the alias created by Player 2. Meanwhile, Player 2 gets nervous about the idea of aliases for primitives; what if a *primitive* (or even an alias) is redefined? So Player 2 creates another alias for *move* – one that will allow him to still move if the move-primitive is redefined, but which is harder for the other player to guess and change. But Player 1 is much more clever than that; she redefines the primitive move-command so that it will put a player inside a *false game loop*. Sure enough, the next time Player 2 tries to move, he falls into the false-prompt and is trapped. However, since control remains with him, he doesn't immediately realize he is trapped; he types the command for *look* and then realizes that no matter what command he gives, he will simply be shown his inventory.

Even with this minimal example, hopefully it is clear how there are many other interesting possibilities for innovative game-play. To highlight a few of those possibilities, consider some additional ways to leverage the potential of fake game-loops.

Players can:

- create their own syntax. That is, without even worrying about creating "traps" for each other, a fake prompt specifies a specific syntax, and can simply function as a "syntactic interface" to the underlying game-commands

- design a fake-prompt so it looks like the "real" prompt. Thus, in cases where it is possible to "escape" from a fake-prompt, it may still take some time for players to *notice* that they are not working within the original game-loop

- implement the fake-prompt so that "trapped" players use one command (e.g., `move`) – but it actually executes a *different* command in the "real" game (e.g., `drop cloak`)

cludes a *robot* in the world when the game begins – and the ability for players to create their own robots. A robot is simply a particular kind of program running inside the game, and it can be instructed to do things so that, for example, players can explore some part of the environment without putting their own characters at risk. We will not discuss the robot further in this paper.

4. DISCUSSION

Once we started using Scheme to develop a particular work, we rapidly started thinking in terms of such things as REPLs ("Read Evaluate Print Loops") – how players could interacted with a shared REPL, how to implement the entire game-world in purely functional terms, and the like. Some of these ideas have already been implemented as part of *SoD* – and some have not.

In some cases, there is no principled or practical reason why such designs cannot work – and, indeed, everything suggests that they can be *interesting* as part of reader/player experience. So, for example, the full design-specification of *SoD* also includes the ability for players to treat larger aspects of the game – such as geographical layout of the world – as running programs that can be modified in various ways; thus, a player could insert an *additional door* next to the door from Foyer to Cloak Room – and create a fake "look-alike" Bar in between the Foyer and the real Bar. Although there is no technical barrier to implementing this functionality, we have not yet completed this feature at the time of writing this paper. Other possibilities also seem interesting: distinctive aspects of recursive *processes* (tail-recursion, tree-recursion, etc.) can form the basis for a number of interesting possibilities, such as the run-time generation of world-elements and geography; and *higher-order procedures* – procedures that can take procedures as arguments and can return procedures as values – suggest such things as "Cat in the Hat"-like agents.

In other cases, the issues are less clear. For example, as of this writing we are still exploring the consequences of allowing players to create their own false-prompts. The challenge, from a game-design perspective, is that a game would probably not be interesting if players were able to "fall into a false-REPL" without being able to get out of it. Although there are a number of reasonable ways to resolve this, we are not yet fully satisfied with any of them. Nonetheless, the potential of such false-prompts seems intuitively worth further work to see if a satisfying resolution can be developed.

Similarly, the ability to redefine game-primitives raises issues of game design. Although this feature is available in standard Scheme implementations, the developer has the ability to reset the system "from scratch." In *SoD*, "resetting the system" is not usually going to be a viable option for players – which leads to some design issues. If a player redefines a game-primitive, such as `move`, so that it is bound to `sleep`, then there is no way for either player to use (or even access) the original `move` – and thus there is a risk they will have no way to control their movement in the game. In our current implementation, we simply leave this as a "challenge" for the players; that is, they need to be aware that they can be trapped by such redefinitions and build appropriate safeguards. Nonetheless, this solution as it stands can still lead to play that simply "stalls", so it is not fully satisfactory. One design alternative is to restrict the rebinding of certain primitives in various ways; another option is to have an "unbind" command.

Another unresolved design issue is related to the specifics of the user-language available to the reader/player. In the

specific case of *SoD*, it happens that the user-language is the same as the implementation language and thus mirrors its specific syntax; this, of course, is not strictly necessary – but it has some interesting advantages and disadvantages. One advantage is the possibility of certain kinds of interesting narrative/game complexity from meta-programming. A disadvantage is that the complexity or unfamiliarity of the language may be a barrier to use.

But perhaps the largest unresolved issue is the degree to which the choice of a particular *programming* – as opposed to *narrative* – paradigm can actually contribute to the development of IF authoring-tools for creating innovative *narrative* works. That is, our work on *SoD* is obviously very much at the game-playing end of the IF spectrum. We took the *Cloak of Darkness* "story" as a given and concentrated on transformations of game-play, rather than transformations of narrative form. Such systems as the one we developed may help IF authors (or players) make more interesting puzzles, problems, and games for each other – but they leave open the questions about the creation of interesting narrative.

Of course, aspects of game and narrative design interact, so player-inventiveness could take place along either game- or narrative-dimensions. For example, players can treat each other as opponents and create puzzles/problems to make it difficult to win the game – or they can treat each other as an "audience," where the emphasis is on the dynamic creation of entertainments/diversions that constitute an interesting and unique shared experience. And, of course, these are not mutually exclusive options. In the best of cases, one can imagine real-time, co-adaptive game and story-creation, as with the way some children's games evolve: the rules, goals, and play emerging as a result of the process of playing itself.

From a wider perspective, the use of particular technical tools by an author/designer can have interesting expressive consequences for the kinds of experience created for readers/players. One could make the case that the difference between the authorial use of pen, typewriter, or word processor contributes to attendant changes in the writing *style* of the works produced. Similarly, it could be said that the invention of three-point perspective in art contributed more than a "simple" technical improvement to visual representation; it may have also contributed to a shift in focus, from painting as "temporal stories" to painting as "snapshots in time." In non-computational literature, authors as diverse as Laurence Stern, Vladimir Nabokov, Peter Handke, Ursula Le Guin, and Italo Calvino have used innovative narrative structures and techniques to create *playful* literary works that engage readers in meditations about questions such as, "what is a story?"

It does seem *possible* that providing players with the power to create in-game programs can facilitate the development of innovative narrative structures. Thus, for *SoD*, a possible narrative question that readers/players could raise is, "what is an *ending*?" But we certainly would not want to make any ambitious claims for our modest work to date – nor even suggest that such player-invention *will* happen. So, although we feel that the ability for players to program as part of play does suggest some interesting narrative possibilities, to the extent that there is any narrative "innovation", it is "emergent", much as it is in systems such as The Sims. The novelty of the narrative experience is "whatever the players make it." Nonetheless, there is much that is suggestive for the interplay between the technical ideas that inform story-creation and the expressive range of story-structures. And independently of whether we have been able to solve any *particular* design problems, we are nonetheless happy to be engaged in this *class* of design problem. That is, our experience as designers/developers is one in which we are working with interesting IF *design* issues.

5. CONCLUSION

In this paper we have emphasized some of the advantages of using different programming paradigms as the basis for IF authoring-tools. *SoD* is a particular example of a work created this way; in this case, it involves creating a game infrastructure that encourages and supports programming by players/readers. Such a model suggests interesting possibilities for designers as well as for players. In particular, designers of such games become meta-designers – programmers who develop programming frameworks for players-as-programmers. Furthermore, for this project we have been *meta-meta programmers*: designing for designers/authors of such programmable IF works, and our experience may also suggest some useful insights for other meta-designers. One of the techniques we used is well-known to programmers familiar with the implementation and design of programming languages, particularly interpreted languages. Namely, the implementation of a *meta-circular interpreter* [1] – or an interpreter implemented in the language which it is intended to interpret.

Readers familiar with EMACS will recognize this as a variation on the EMACS implementation strategy. EMACS is (among other things) an editor that supports the development of computer-programs – and is largely implemented in such a way that the EMACS editor can be used to modify and extend the EMACS editor *itself*. That is, EMACS is built in a language where a minimal "core" of that language is implemented in some lower-level language – and *the rest of the language* is implemented with that "core." Most of the implementation language, then, exists as a modifiable "extension language" available to users of the editor. Providing extension-languages with applications is very useful when users of the applications may want to add their own functionality to those applications. In the case of EMACS, the extension-language happens to be the same as the implementation-language of the application and most of the application's actual source-code can be modified by users of the application; this allows users to customize the application in quite radical ways.

Techniques like this can also be very useful in the development of new programming languages; they make it easy to experiment with variations on the *designed language itself*. And this concept – of making it easy to experiment with language design – seems like a powerful idea for the development of both IF languages and IF works. This involves more than simply adding this or that feature for IF designers ("more rooms" or "dynamic reconfiguration of world-layout"), rather, it supports immediate changes to aspects of the IF language itself. This is obviously helpful in situations where one is experimenting with the possibility of IF works that provide programming capabilities to players – but it also seems more generally useful as a model for designers and users of IF languages.

This is an example of one of our original concerns, reformulated in terms of the *designers of IF authoring tools* rather than in terms of *authors of IF*: rather than simply making theories about appropriate and interesting tools for IF authors, as developers of IF authoring tools we are building and testing different possibilities.

With regard to this overall goal of increasing the expressiveness of IF authoring-tools, it is important to stress that it is not Scheme nor functional programming in particular that is our main interest. We could have chosen to explore possible design consequences of developing an IF language within a constraint-based or connectionist or production-system paradigm. Similarly, our work to date takes *Cloak of Darkness* as a self-imposed constraint to focus the development of a particular IF authoring-language and to explore ways in which a very limited IF "narrative" could be transformed with it. It will be interesting to explore whether and how we can begin to develop more interesting IF narratives with our IF authoring-language.

5.1 Directions for Future Research

In closing, we would like to indicate two classes of interesting future work: elaborating specific features of *SoD* and tool/language features for authoring such systems – and transforming such work into useful IF theories.

As an example of future tool-design, it would be interesting to extend the existing implementation of *SoD* to support runtime graphical/multimedia transformations. We can imagine many nice extensions to this basic concept of user-programming, allowing players to, for example, programmatically control and change how aspects of the story/world appear or sound.

Similarly, with *SoD* we have not yet explored some obvious concepts that arises from using a language such as Scheme: self-reference, self-modification, the use of *continuations* to manipulate temporal flow, and the like. And such explorations may actually contribute to a better understanding of potential interplay between game and narrative structure. Films such as Christopher Nolan's *Memento* and works of fiction such as Calvino's *If On a Winter's Night a Traveler* are self-referential – but there is also a particular semantic dimension to them that is interesting from the perspective of narrative theory. Namely, the *content* of such works explicitly references their *narrative techniques* – and the narrative techniques explicitly parallel the content. Some of the more interesting works of IF also have this quality. *Photopia* by Adam Cadre and *The Dr. K–Project* by Brandon Rickman, for example, each play with expectations about *interactivity* in IF; in *Photopia*, players/readers cannot actually make choices of consequence – and the story itself involves questions about free will and choice. The quality of such "self-reflection" may be an indicator of progress in the emergence of new forms; it may be no coincidence that *Don Quixote*, widely considered the first Western novel, is an extended meditation on the nature of literature. Such stories, in a very real sense, *become what they are about*. And such examples – where structure, technique, and content start to co-inform each other – suggest ways in which IF tool-design that makes use of concepts from different programming paradigms could support the expressive efforts of authors of innovative IF works.

Given what we have said so far about the importance of tools for *creating* innovative IF, it may be surprising that we are interested in the potential of such work to also inform IF *theory*. Nonetheless, for us there is no contradiction: the work on tools and example works can inform the elaboration of new theory – and such theory can, of course, inform further experimental work. As an example of a more theoretical concern, consider that the *role of the reader/player* in typical IF is either largely conceived as one of *narrative decoder* or *game-player/puzzle-solver*. But our work to date on *SoD* suggests a third possible role – one closer to Barthes' notion of the *reader/critic as producer*, rather than simply as a *narrative consumer*. In an IF work in which the act of "reading" blurs with the act of "programming," the reader can become literally (!) productive in practice.

Indeed, in such a system, the reader/player's *dispositions* and the disposition of the text/game may even become co-constructive in interesting ways.

References

[1] H. Abelson and G. J. S. with Julie Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, 1996.

[2] A. Bapat, J. Wösch, K. Aberer, and J. M. Haake. Hyperstorm: an extensible object-oriented hypermedia engine. In *HYPERTEXT '96: Proceedings of the the seventh ACM conference on Hypertext*, pages 203–214. ACM Press, 1996.

[3] J. D. Bolter and M. Joyce. Hypertext and creative writing. In *HYPERTEXT '87: Proceeding of the ACM conference on Hypertext*, pages 41–50. ACM Press, 1987.

[4] C. Crawford. *Chris Crawford on Interactive Storytelling*. New Riders, Berkeley, CA, 2005.

[5] S. Donikian and J.-N. Portugal. Writing interactive fiction scenarii with dramachina. In *TIDSE 2004: Technologies for Interactive Digital Storytelling and Entertainment, Second International Conference, Darmstadt, Germany, June 24-26, 2004*, pages 101–112, 2004.

[6] S. Göbel, N. Braun, U. Spierling, J. Dechau, and H. Diener, editors. *TIDSE 2003: Technologies for Interactive Digital Storytelling and Entertainment, First International Conference, Darmstadt, Germany, June 24–26, 2003*. Fraunhofer IRB Verlag, 2003.

[7] G. P. Landow. *Hypertext: The Convergence of Contemporary Critical Theory and Technology*. The Johns Hopkins University Press, Baltimore, MD, 1992.

[8] M. Mateas and P. Sengers, editors. *Narrative Intelligence*. John Benjamins, Amsterdam, 2003.

[9] K. Perlin and A. Goldberg. Improv: a system for scripting interactive actors in virtual worlds. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 205–216. ACM Press, 1996.

[10] C. S. Pinhanez, J. W. Davis, S. Intille, M. P. Johnson, A. D. Wilson, A. F.Bobick, and B. Blumberg. Physically interactive story environments. *IBM Systems Journal*, 39(3–4):438–455, July 2000.

[11] M. Riedl, C. J. Saretto, and R. M. Young. Managing interaction between users and agents in a multi-agent storytelling environment. In *AAMAS '03: Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 741–748. ACM Press, 2003.

[12] M. M. A. Stern. Facade: An experiment in building a fully-realized interactive drama. In *Game Developer's Conference: Game Design Track*, San Jose, March 2003.

[13] P. T. Zellweger, A. Mangen, and P. Newman. Reading and writing fluid hypertext narratives. In *HYPERTEXT '02: Proceedings of the thirteenth ACM conference on Hypertext and hypermedia*, pages 45–54. ACM Press, 2002.

Player 1	Player 2
FOYER OF THE OPERA HOUSE You are standing in a spacious hall, splendidly decorated in red and gold, with glittering chandeliers overhead. The entrance from the street is to the north, and there are doorways south and west. > (inventory) Inventory: 1: blue velvet cloak	FOYER OF THE OPERA HOUSE You are standing in a spacious hall, splendidly decorated in red and gold, with glittering chandeliers overhead. The entrance from the street is to the north, and there are doorways south and west. > (sleep) Time passes ...
> (move 'west) CLOAKROOM The walls of this small room were clearly once lined with hooks, though now only one remains. The exit is a door to the east.	> (define m move) ok > (m 'south) DARKNESS The bar, it is pitch black in here.
> (m 'east) FOYER OF THE OPERA HOUSE	> (define my:move move) ok > (my:move 'north) FOYER OF THE OPERA HOUSE

> (define (move arg)
  (read)
  (display "Trapped: ")
  (inventory)
  (newline)
  (move arg))
 	
ok
 	
> (m 'south)
 	
DARKNESS
 	
The bar, it is pitch black in here.
	> (move 'east)
	
	> (look)
	
	Trapped:
	Inventory:
	  1: blue velvet cloak

Figure 2: ***Scheme of Darkness*****, sample game**