

# “Causal Normalisation: A Methodology for Coherent Story Logic Design in Computer Role-Playing Games”

Craig A. Lindley and Mirjam Eladhari

Zero Game Studio  
The Interactive Institute  
Skeppsbron 24  
SE-621 57 Visby, Sweden  
{craig.lindley,mirjam.eladhari}@interactiveinstitute.se

**Abstract.** A common experience in playing computer role-playing games, adventure games, and action games, is to move through a complex environment only to discover that a quest cannot be completed, a barrier cannot be passed, or a goal cannot be achieved without reloading an earlier game state and trying different paths through the story. This is typically an unanticipated side effect caused by the player having moved through a sequence of actions or a pathway different from that anticipated by the game designers. Analogous side effects can be observed in traditional software engineering, referred to as data coupling and control coupling, in database design, in terms of unnormalised relations, and in knowledge base design, in terms of unnormalised truth-functional dependencies between declarative rules. In all cases, good design is a matter of minimising functional dependencies, and therefore coupling relationships, between different parts of the system structures, and deriving system design from the minimised dependency relationships. We propose a story logic design methodology, referred to as causal normalisation, that minimises some forms of causal functional dependency within story logics and therefore eliminates some unintended forms of causal coupling. This can reduce the kind of unexpected dead ends in gameplay that lead to player perceptions of poor game design. Normalisation may not be enough, however. Extending the principle of minimal coupling, we propose an object-oriented approach to story logic, and relate this to principles of normalisation and game architecture.

## Introduction

The study of games and gameplay has historically been concerned with the study of competitive systems, associated with economic theory more strongly than with play. Traditional board games and puzzle games typically model competitive situations in a very abstract way, involving little or no story context, game world, or characterization. It is only with the advent of computer games that the distinctions between games/gameplay and narrative have become unclear, and the study of games has shifted focus more strongly towards games as a type of fiction. Computer games span a range of forms, varying from the highly narrative, to the highly non-narrative. This range of perspectives, from the ludological to the narratological, is depicted on

Figure 1. At the ludological extreme are computer implementations of traditional board games, and abstract game forms that rely upon the active dynamics of a computer implementation, but have little or no function in terms of representing a fictional world. At the narratological extreme are highly story-oriented productions, from multipath movies to hypertext stories and adventures.

The more dominant computer game forms lie in a continuum between these extremes, using different approaches for the integration of narrative and patterned gameplay. This may lead to a perceived tension between gameplay and story in computer role-playing games (see, for example, Aarseth, 1997), although this ultimately amounts to a matter of style and taste; players will gravitate to the games that satisfy their preferences in terms of narrative framing and its relation to the core gameplay experience (or the *gameplay gestalt*, as an essentially non-narrative interaction pattern, Lindley 2002). There are game forms more concerned with simulation, or realising multi-user, on-line avatar worlds, that strain the definition of games, although similar tensions between game play and story also occur in these systems.

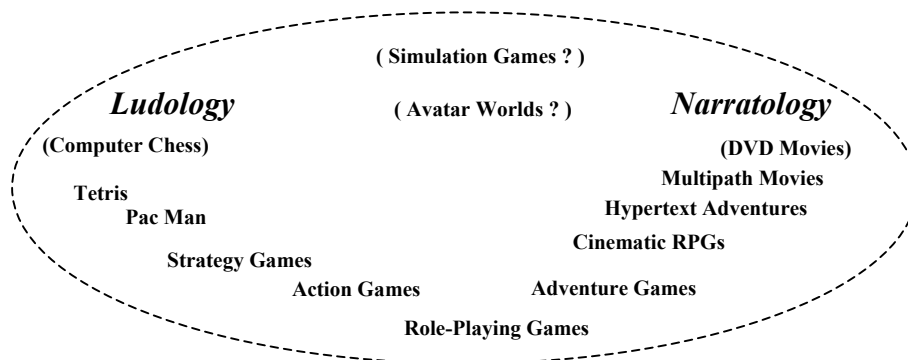


Fig. 1. Games fall within a continuum from the ludological to the narratological

More consistently frustrating for players than the gameplay/narrative tension is the experience of moving through a rich game world, completing the tasks, meeting the challenges, completing the quests, etc., only to come to a point that is unpassable since some unknown critical action has not been performed at a previous point in the game. This may result in the player needing to experiment with reloading past game states in order to try to discover the “correct” sequence that needed to be completed, restarting the game, or floundering around with no idea of what went wrong or how to move forward. If the frustration level is too high, the player will stop playing the game, leaving her with the impression that it was too hard, impossible to understand, too tedious to resolve, and/or badly designed. This is a problem of story logic. Story logic becomes more explicit in the movement from the ludological pole towards the narratological pole of Figure 1. However, story logic problems are most prevalent in the intermediate zones, where story design is obscured by gameplay and simulation.

In this paper we consider the extent to which problems in story logic can be attributed to design characteristics that can be analysed in software engineering terms, ie. in

terms of dependency relationships among story elements. We present a simple example of a stereotypical game quest, together with two undesirable outcomes that have actually been experienced by the authors in real game play. We present a semi-formalised representation of the causal relationships involved in the quest and the problematic situations. We then review the concepts of coupling, functional dependency, and normalisation theory as they have been used in the history of software engineering methodologies, and explore the applicability of these concepts to story logics. A number of principles for normal forms for story logics are presented. We also consider the applicability of object-oriented concepts to storytelling, and what this means in terms of system architectures for games.

## **The Dead End – Errors in Game Logic**

The forms of games in which stories dominate the player experience are branching narratives based upon a hypertext model (eg. multi-path movies, <http://www.brilliantdigital.com/solutions/movies/>); at the other extreme, strong gameplay can be experienced in very abstract games having no story at all, such as traditional board and puzzle games. Problems in game logic of the kind discussed here lie in intermediate forms, where game designers wish to impose a specific series of plot points in order to create particular story structures, but game players are given high levels of freedom in interaction, especially in relation to the exploration of the landscapes and architectures of a game world.

A common strategy for imposing a specific story sequence within a highly interactive game is to make progress in the game conditional upon completing a specific sequence of actions or plot points. This is where design problems may arise. Consider, for example, the following clichéd scenario. The player plays the part of a fantasy protagonist (the player character, or PC) moving through a medieval world inhabited by various helpful or enemy non-player characters (NPCs). The designers have created a quest: an ailing wizard will give the player a key to an underground cave system in return for killing an old enemy dragon that the wizard has failed to destroy in time before his own death, and which therefore now threatens the local town. This is programmed into the game. However, as a function of the virtual geography of the game, the player's interactive possibilities for traversing this geography, and the way the quest is imposed upon the player, several story outcomes are possible. First, the outcome intended by the designers:

1. the player meets the wizard and is given the quest. The player follows the wizard's instructions, finds, battles and defeats the dragon, returns its head to the wizard, and is rewarded with the key. The player can now continue in the game by seeking and entering the underground cave system to further her higher level quest.

As a simple example of a design problem, however, we consider the case when the player has enough freedom in the environment to go to the dragon's lair before going

to the wizard's lair. This occurs in part due to the game designers attempts to simulate a world, since one solution for the dependency problem (generally undesirable for players) is to restrict freedom of movement in the world to enforce the required sequence of events. One design solution for imposing the intended story without restricting freedom of movement is to not instantiate the dragon until the wizard has been encountered. This leads to the following possible outcome:

2. the player goes into an empty lair (no dragon yet). The player goes on to receive the wizard's instructions. The player is now looking for a dragon in a lair, but does not go back to the lair because it was previously found to be empty. The player searches through all reachable but previously unexplored parts of the terrain. No dragon is found. Either the player must revisit all previously visited areas of the map just in case one of them was the lair which is now by chance inhabited, or will give up, having no options to go anywhere new, and not understanding why the dragon is not to be found.

To avoid this, designers allow the dragon to be in its lair before the player character visits the wizard, leading to another possible outcome:

3. the player goes into the dragon's lair, battles and defeats the dragon. The player then goes on to meet the wizard and is given the quest. However, the quest cannot be completed, because the dragon no longer exists. The player must reload a game state prior to the point of defeating the dragon, and go through the battle again, this time after visiting the wizard. If no suitable state has been saved, the player must restart the game, or stop playing.

Of course, there are solutions that avoid these outcomes. For example, to avoid outcome 2, the wizard can explain where the lair is, and the designers can hope that this can be related to the player's memory of the lair if it has already been visited. This can however detract from the fun element of finding the lair as part of the quest, and also raises the question of why the player didn't run into the dragon along the long and winding route from the lair to the wizard. The solution violates the expected existential logic of the world for the sake of a specific story sequence. Outcome 3 can be avoided by having the wizard reward the player's action of killing the dragon even though the action was performed before the player was instructed to do it, so it is no longer necessarily a quest. This is a matter of weakening the imposition of the designer's desired story sequence, for the sake of a more plausible simulation of a world.

While these solutions are possible, they and the situation leading to them raise the question of whether there is a more general and coherent method for understanding and resolving this kind of problem in story logic. Here we propose two methods; firstly, we consider the analysis of causal dependencies in the game logic, including the notion of *causal coupling*, and a design methodology based upon the minimisation of causal coupling by *causal normalisation*. This approach is appropriate when specific story structures (such as quests) are desired as an intrinsic part of the game form. The second approach, that of object oriented storytelling, is desirable when the

world is to function more as a simulation, in which stories are an emergent and retrospective phenomenon.

## Causal Modelling for Game Logics

The story example above can be represented in the following way. We use the notation:

$E1(\text{P meets W and receives Q}) \rightarrow E2(\text{P goes to L})$

to represent a causal relationship, where:

E1 and E2 denote events 1 and 2, respectively

P refers to the player

W refers to the wizard

Q refers to the quest instruction

L refers to the dragon's lair

$\rightarrow$  is a causal relationship, where the event(s) on the left hand side of the arrow causes the event on the right hand side of the arrow.

We have not completely formalised this notation, nor adopted an existing causal logic, but find this level of formalisation sufficient for the analysis presented here, ie. as a tool for the analysis of patterns of causal dependency.

Using this notation, we present outcome 1 above in terms of the following sequence of causal dependencies desired by the game designers.

Sequence 1:

$E1(\text{P meets W and receives Q})$   
 $\rightarrow E2(\text{P goes to L})$  where L denotes the dragon's lair  
 $\rightarrow E3(\text{P meets D})$  where D denotes the dragon  
 $\rightarrow E4(\text{P defeats D})$   
 $\rightarrow E5(\text{P returns victoriously to W})$   
 $\rightarrow E6(\text{P receives R})$  where R denotes the reward

A crucial issue in game design is whether or not to impose these kinds of causal relationships as rules that the player must obey. This becomes very complex, since a decision to impose causal rules raises the need for desirable formal properties, such as soundness, completeness, decidability and consistency (see Frost, 1986). The undesirable outcomes 2 and 3 above result from the lack of these properties for the causal system expressed in Sequence 1. For instance, the system is incomplete in the sense that E2 can be true without being derived from (or caused by) E1. The presence of the player as an active causal agent in the game world, and the function of that world as a simulation, make it impossible to formalise all possible simulated causal

relationships in that world, so a formal approach to proving desirable behaviour is generally not feasible.

Examining outcome 2 above, in which the user encounters the lair without the dragon prior to encountering the wizard, we find the causal sequence:

Sequence 2:  
E2  
- > E1  
- > confusion!

Outcome 3 involves the sequence:

Sequence 3:  
E2  
- > E3  
- > E4  
- > E1  
- > E2  
- > E5  
- > dead end!

Since these problems arise from undesirable patterns of causal dependency, it may be feasible to apply systematic methods from software engineering practice, based upon dependency analysis, as an aid to story logic design.

## **Coupling, Dependency, and Normalisation in Software Engineering**

The analysis of dependencies underlies methodologies for system development within a variety of programming and development paradigms. This includes structured development (analysis and design) for procedural software systems (Yourdon and Constantine, 1979), normalisation of relational database systems (Codd 1970, 1971, 1972; Date 1981), and normalisation of rule-based knowledge systems (Debenham, 1989, 1998). Structured development for software systems is based upon an analysis of the data flow relationships within an application, as captured by hierarchical data flow diagrams (DFDs; see De Marco 1978, Gane and Sarson 1979). A data flow diagram is a representation of the data within a system, and how data flows between different transforming processes. Structured software development methodology has traditionally used DFDs to represent data flow as part of the analysis of a system, and the resulting DFDs have then been used as a basis for hierarchically defining program modules. In developing this approach, Yourdon and Constantine (1979) articulate the concept of *coupling*, as the degree to which one functional module of a system must know about another, which then amounts to the likelihood that modifications to one module will effect the operation of another in some way. Coupling can be further classified into *data* coupling and *control* coupling, where data coupling involves a

data dependency between modules (modifying a data value in one module changes the data outputs of another), and control coupling involves a control dependency (the behaviour of one module influences the control sequencing of another). A good structured design amounts to creating system with a minimum of coupling between modules, so that future modifications to a module will have a minimum impact upon the operation of the rest of the system. Structured analysis and design techniques focus on data flow relationships, and seek to minimise data functional dependencies between modules by defining systems having a structure that reflects data dependency.

Database normalisation involves constructing relations for relational databases that reflect the functional dependencies within the data domains. A functional relation from a domain A to a domain B means that a value within domain A uniquely determines a value within domain B; values within domain B can have more than one determinant in A, but each value in A has only one dependent value in B. Database normalisation is a process of eliminating redundancy and inconsistent dependencies within relational database designs by following the patterns of functional dependency within the data domains (see Date, 1981). This can be seen to be a very similar process to the minimisation of coupling in structured analysis and design (or identical at an abstract level), the difference being that in pure database systems, values are explicitly represented rather than being calculated dynamically.

Normalisation theory is extended into rule base systems by Debenham (1989, 1998), in this case dealing with the same or similar kinds of functional dependencies, but expressed in terms of abstract declarative relations, instead of database tuples. These dependencies are truth-functional dependencies, and normalisation amounts to the minimisation of truth-functional coupling. A simple example is the separation of repetitive premise subsets into distinct rules, analogous to Codd's first normal form for database systems. For example, consider the simple propositional rules:

Rule 1:  
A, B, C, D, E    :-    F

Rule 2:  
G, H, C, D, E    :-    I

where capitalised letters represent simple propositions, and :- represents logical implication. The occurrence of the subset of premises {C, D, E} in both rules suggests an interdependency between the propositions within the subset. This creates an update hazard, since any change to this interdependent set must be reflected everywhere that it occurs. Rules 1 and 2 are therefore truth-functionally coupled in the sense that the {C, D, E} subset represents a common meaning, which becomes ambiguous if the expression of that meaning becomes inconsistent in different rules. To avoid this, the rules can be normalized by extracting the subset as a new rule, and replacing the subset by the head of the new rule in rules 1 and 2, giving the new rule set:

Rule 1:  
 A, B, J            :-        F

Rule 2:  
 G, H, J            :-        I

Rule 3:  
 C, D, E :-        J

The meaning of the {C, D, E} subset is now encapsulated within Rule 3, and changes to the subset only have to be made in one place. As with structured software design and database normalization, the representation structures reflect the functional dependencies within the system.

Object-oriented software development methodologies (see Booch, 1994) have superseded many of the earlier methodologies, as a more coherent and universal method of addressing the standing issues of minimizing modular coupling and providing a principled approach to system development. Object-based approaches provide a consistent methodology through all phases of software development, since objects identified during analysis may provide the foundation for objects in the design and implementation of systems. An object encapsulates both data and control, and provides what should be well-defined interfaces through which other modules can use their functionality. Object-based systems typically also use the concept of inheritance, allowing system constructs to be defined as classes at various levels of abstraction, with lower level constructs inheriting features, data, and/or functions (methods) from higher abstraction levels. An object is then an instance of a class, having it's own internal data (state information), and interfaces defined as methods by which other objects can interact with it. Ideally, a system composed of a set of interacting objects has minimal control and data coupling between its elements.

In the next section we examine the meaning of principles of dependency analysis for story logics. The issue of object-orientation in story structure is examined in the section after that.

## **Causal Normalisation For Games**

Examination of sequence 1 together with outcome sequences 2 and 3 shows that these outcomes result from dependent and independent relationships that are not clearly represented in Sequence 1. In particular, outcome 2 results from a dependency between E1 and E3. That is, the player can only meet the dragon if she has first encountered the wizard. Outcome 3 results from a dependency between E4, E5 and E1; the player can only return to the wizard after killing the dragon and receive a reward if the wizard has been visited before the dragon was killed. In both cases, the ability to enter the sequence at E2 undermines the intended story logic.

This kind of causal influence resembles control and data coupling phenomena in software engineering, and unnormalised relationships in databases and rule base systems. In all cases, there are dependencies that cut across the intended, explicit, or modelled dependencies of the system. For story logics we can refer to this as *causal coupling*, informally understood as a causal relationship that is excluded from a high level causal model of the story logic. If causal coupling is ignored, sequence 1 could be represented by a sequence of separate causal steps, as follows.

Sequence 4:

E1(P meets W and receives Q)	->	E2(P goes to L)
E2(P goes to L)	->	E3(P meets D)
E3(P meets D)	->	E4(P defeats D)
E4(P defeats D)	->	E5(P returns victoriously to W)
E5(P returns victoriously to W)	->	E6(P receives R)

If each step is treated as a causal rule within the system, then the occurrence of a cause event must be followed by the occurrence of an effect event. This allows sequence 1 to be sidestepped to different degrees, due to the nature of the game world as a simulation in which the traversal of the world by the player character, or the player character's affect within the world (eg. via magic), is not constrained in terms of this causal rule set. For instance, the player might remotely defeat the dragon by magical or other indirect means, without ever having met either the wizard or the dragon. Then E4 is satisfied, and by the steps E4 -> E5 and E5 -> E6, the player receives the reward from the wizard.

If the designers wish to impose the strategies that lead to outcomes 2 and 3, we can explicitly represent what were the hidden dependencies between E1, E3, and E5 in sequence 1 by modifying the causal steps of sequence 4 as follows :

Sequence 5:

E1(P meets W and receives Q)	->	E2(P goes to L)
E1(P meets W and receives Q) and E2(P goes to L)	->	E3(P meets D)
E3(P meets D)	->	E4(P defeats D)
E1(P meets W and receives Q) and E4(P defeats D)	->	E5(P returns victoriously to W)
E5(P returns victoriously to W)	->	E6(P receives R)

Now it is possible to see that the causal relations expressed within the second rule include a hidden relation within the causes analogous to that addressed by Boyce-Codd Normal Form (BCNF) in database theory (see Date, 1981). This is because E3 is caused by E1 and E2, while E2 is an effect of E1. The hidden dependency creates precisely the kind of anomaly observed in outcome 2, that if E2 occurs without E1, there is no specified outcome. Similarly in rule 4, if E4 occurs without E1, there is no specified outcome, although in this case there are no dependencies between E1 and E4.

Modelling the previously hidden dependencies clarifies the existence of undesirable game states. It can also be asked if there is a methodology analogous to normalisation that can be applied to causal models of this kind that might prevent or help to prevent

these anomalies from occurring. Applying the principle of BCNF to the second causal step of sequence 2, we could break it down into the first two separate steps of sequence 4:

E1(P meets W and receives Q)	->	E2(P goes to L)
E2(P goes to L)	->	E3(P meets D)

These relationships are normalised in a form analogous to BCNF, eliminating interdependencies between the causes within a single relationship. Now we are back in the situation of no longer imposing the logic that leads to our earlier outcome 2. It appears that the imposition of a desired story sequence creates the unnormalised story structure responsible for the undesirable outcome.

This analysis suggests that it may be possible to define a general set of normal forms for the causal relationships in story logics. Assuming a representation of causal relationships that links a set of causes to a specific effect, such normal forms for story logics should at least:

- extract recurrent subsets of causes representing independent events as separate cause-event relations (an analog of Codd's first normal form for relational databases)
- eliminate irrelevant causes from cause sets (an analog of Codd's second normal form)
- separate multiple effects of a common set of causes into multiple relations, one for each effect (an analog of Codd's fourth and fifth normal forms)
- eliminate interdependencies between causes within any single relation (an analog of BCNF)

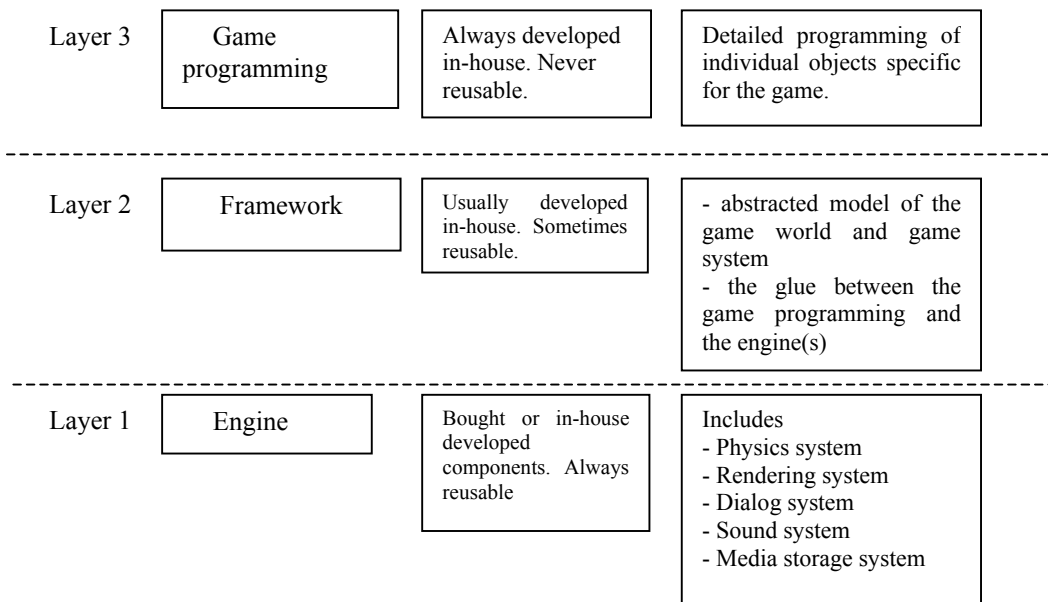
Developing these ideas into a more precise, extensive, and formalised list of normal forms for causal relations is beyond the scope of this paper. Such a task will depend upon settling upon a specific representation for story logics. This should be able to be done for any explicit representation of causal dependencies in stories, and applying the above normal forms to the analysis of those dependencies. Using causal normalisation, it should be possible to eliminate story logic anomalies for games in which the story logic covers all possible traversals of the game world. These are the games close to the narratological pole of Figure 1.

## **Normalisation Methodology and System Architecture**

Database normalisation theory derives from the relational formalisation of database functionality. Relational databases are designed in accordance with this model, so the abstract methodology has a deep relationship to the operational semantics of a relational database. Applying a normalisation method to the story logic of a computer game requires a similar mapping from a representation that is convenient to normalise, to the semantics of that representation in terms of the simulated events and

player experiences of the game world. What is required, therefore, is a mapping between different levels of interpretation of “the game”. For a story-driven computer game there are three levels that internally form text layers and structures:

1. The *code level*, consisting of engines, a game framework and game programming. These together define the mechanics, the virtual geographical structure of the game world, and the conditions for the overall story and its deep structure.
2. The *narrative level*, consisting of the overall story, deep structure and the specific story carrying objects, which in turn can manifest the story, possible side quests and internally independent stories.
3. The *discourse level*, consisting of the sequential order that is created between the parts of the narrative simultaneously with the players movement through the game. It is at this level that the surface structure of a game text can be monitored.



**Fig. 2.** Layers of text in the code level in computer games.

In these terms, given a game engine constituting the code level, the narrative level may be a primary concern for the game designer (who might implement the level as data to be inserted within the code level). The discourse level is the game play experience and the experience of the game as a story on the part of a player. Causal normalisation is a narrative level methodology to help to ensure that the data entering the code level creates a coherent story experience at the discourse level. Effective normalization must be treated as an issue of defining a coherent and useable narrative level methodology together with a clear migration path to the semantics of narrative representation within the code level, just as a normalized logical relational database model has a clear mapping to table structures within a relational database

(notwithstanding pragmatic variations in implementation). Before this is possible, it is necessary to devise clear representations for the narrative level and its semantics.

In general, what we have referred to here as the code level of a game can itself be subdivided into three levels, as shown in Figure 2. The lowest layer of the code level is the engine, consisting of very general functions, such as the rendering system interface, animation interfaces, collision detection, terrain or portal management, a dialog system, and media storage and access. Above this is the framework for the game, which is the level of abstract representations of game structures, such as game agent classes, behaviour controller classes, an event management system, and a communication (ie. message passing) system. The engine may be general across many game genres, while the framework may be more genre-specific. On top of these levels is the specific game programming, consisting mostly of data and instance definitions for realizing a specific game. These layers together present the media that to the player is the game.

How this architecture is built and where the borders are between the layers is different from game to game, from developer to developer, and from genre to genre; it also depends upon the technical platforms and environments of the game.

## **Object-Oriented Storytelling and the Minimisation of Causal Coupling**

For story logics within highly interactive game worlds, where issues of story do not totally dominate the world simulation functions of a game, the concept of normalisation is not as clear as, for example, the case of database systems. Within these worlds, story logics are generally not complex enough to justify a full causal logic, and story structures are often sparse in relation to the size of a game world and the overall cognitive density of the gameplay experience. It is in this kind of world that cases like that of rule 4 of sequence 5 above cause a problem beyond the scope of normalisation. In this case, there is a straightforward stipulation that the player cannot receive the reward without visiting the wizard *before* killing the dragon. This may be the designer's interpretation of the personality of the wizard; the player must act as desired, or miss out on the prize that will unlock unexplored areas of the game world. In general this would be a perverse and undesirable discovery for the player, and we need a better method for reducing such chains of dependency for more flexible game play. This can be accomplished by pursuing object-oriented storytelling, as a strategy for designing game entities in terms of story potential, rather than imposing causal dependencies.

### **Object oriented storytelling**

Object oriented storytelling is an approach in which all objects in the world have integrity and contain their own stories, functions, possible developments, and possible

responses to actions conducted by other objects that influence them. That an object has integrity means that the information available in the object is only available through the object, and all information retrieval or data access is implemented by objects. For object oriented storytelling, this may function as follows.

If a player object, controlled by the player, comes close to a non player character (NPC) object in the game, communication between the player and the NPC is partly defined by the characteristics of the player object, and partly by the characteristics of the NPC object. Depending upon what it has been through earlier in the game, the player object can ask questions governed by the events that have become the history (recorded past) of the player object. The information that the NPC object in its current state can give is dependent on its own history, the location in which the player object encounters it, the time of day it is in the game, etc.. Thus the content of the dialogue and amount of information transferred from the NPC is dependent on a combination of conditions emerging from the meeting between the two objects. By maintaining the integrity of the object, false or confusing causalities need not occur. Actions of the NPC that may be undesirable from an overall story perspective can be avoided by encapsulating knowledge within appropriate objects. For instance, the existence or not of a dragon does not need to be conditioned upon remote interactions that have nothing directly to do with the dragon. An isolated action or state variable that the player object carries can directly correspond with an opportunity to activate a specific response.

This situation corresponds exactly to that discussed in the case of sequence 4 above, and requires the designers to abandon the imposition of prespecified sequences; if the player character goes to the wizard after killing the dragon and without having received the quest, she is nevertheless entitled to the reward. This follows from the simulated intent of the wizard to reward the act of killing the dragon with a key, without making knowledge of this intent a cause within the game world.

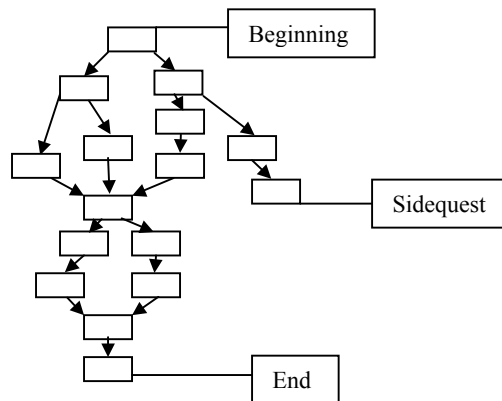
This kind of object oriented approach means that it is unnecessary to create an overall story structure having a large number of conditions for which the internal relations must be correct in order to activate the specific response. The system governing the story logic will be more immune to the kind of causal logic problems discussed above. The advantage of this from a story perspective is that it is possible to construct an NPC and define exactly how it should behave according to its characteristics, the operations that can be performed on it, and the internal conditions set for releasing information to a player character. The advantage from a gameplay perspective is that this NPC and other objects will seem more natural and intelligent, since there are no false casual relations conditioning their behaviour.

### **Mixed Forms: Object-Oriented Storytelling and the Imposed Quest**

In a highly simulation based game, built according to principles of object-oriented storytelling, a quest or a story becomes a history of interaction, as suggested by Oliver

(2001). Rimmon-Kenan (1998) derives a definition of story from Genett's (1983) concept of *histoire*, but stresses the chronological aspect of the term: "Story' designates the narrated events, abstracted from their disposition in the text and reconstructed in their chronological order, together with the participants in these events." In a story-driven computer game in the genre of the adventure game, and within the high level structure of other genres such as role playing and action games, there is a chronological order in which the events occur within a particular player's experience. This order depends, however, on the nature and implementation of the story generating structures at the code level, which are usually not strictly linear and contain more or less possible chronological variations in what Anna Gunder (1999) would call the omnistories. The omnistories in turn contain all possible real stories, that is, all possible chronologically ordered sequences of events. This must be regarded as a combination of possible variations both in the chronological ordering of events, and in the necessity or contingency of occurrence of events.

In a simulation based game, the omnistories are vast and effectively unknowable. Massively multiplayer on-line roleplaying games (MMORPGs), for example, contain unlimited story potential. However, a role for a player character is still typically understood in narrative terms, providing purposes for the character in the game world, generally in the form of quests. A picture of the events that can occur and their possible causal order in a story between a start and an ending might look like the directed network structure shown in Figure 3:



**Fig. 3.** A network structure depicting causal relations between events.

Each square in Figure 3 represents an event. The arrows may represent causal relations between events. In order for an event to happen, the events that are represented by the boxes that have an immediate above connection must have happened. In a model like this, all existing causal relations are important. Thus there are only three existing hierarchical levels:

- 1) cogent relations leading to the end of the story,
- 2) relations that are only cogent for experiencing a sidequest, and

3) events that are not cogent at all and thus not represented in the model.

In these terms, the player's freedom to move their character beyond the structures of the predefined narrative reflect the simulation functions of the game and game world, representing a realm of non-cogent events from the perspective of the designed narrative patterns. Causal normalization is applicable to the narrative model, irrespectively of the non-cogent events. But to such a simple causal map must be added the complexity of the contingency or necessity of causal relations, and relations of joint sufficiency and joint necessity. Only then can a causal map represent possible variations both in the need for and order of occurrence of causally related events. This greatly complicates both the design process of narrative structures, and the processes of story normalization, suggesting that for simulation based worlds, object-oriented storytelling methods are much easier to handle.

In a game that uses object oriented storytelling, a high level narrative model could be interesting as tool for planning possible story experiences (or as a tool for analyzing the game). But any such plan should only be regarded as a picture of a subset of story experiences possible within the game world; it should not be imposed upon the player or specified as an *a priori* set of dependencies between game objects.

## Conclusion

Problems of story logic encountered in computer game play are a consequence of a lack of coherent game development methodologies. As discussed in this paper, the problem of defining a coherent game development methodology can draw from principles of software engineering. However, developing complete solutions must involve the development of production environments in which clear methodological principles have a coherent translation into designs and implementations that preserve the qualities of good designs. For games with highly constrained narrative possibilities, causal (or story logic) normalization provides a methodology for avoiding dead ends or confusing situations in stories. For highly interactive, simulation-based game worlds, however, it appears that the idea of imposing predefined story sequences, even branching sequences, must be largely, if not entirely, abandoned. Instead, we require object-oriented methods for encapsulating interesting behaviour and states of game entities constituting a deep and nonsequential structure of story semantics.

## References

- Aarseth E. J. Cybertext: Perspectives on Ergodic Literature, The Johns Hopkins University Press, 1997.
- Booch G. 1994 Object-Oriented Analysis And Design With Applications, 2nd Ed. Benjamin Cummings.

- Codd E. F. "A Relational Model of Data for Large Shared Data Banks", *Comm. ACM* 13 (6), June 1970, pp. 377-387.
- Codd E. F. "Normalized Data Base Structure: A Brief Tutorial", ACM SIGFIDET Workshop on Data Description, Access, and Control, Nov. 11-12, 1971, San Diego, California, E.F. Codd and A.L. Dean (eds.).
- Codd E. F. "Further Normalization of the Data Base Relational Model", R. Rustin (ed.), *Data Base Systems (Courant Computer Science Symposia 6)*, Prentice-Hall, 1972. Also IBM Research Report RJ909.
- Date C. J. 1981 *An Introduction to Database Systems* (third edition), Addison-Wesley.
- Debenham J. K. 1989 *Knowledge Systems Design*, Prentice Hall Advances in Computer Science Series.
- Debenham J. K. 1998 *Knowledge Engineering: Unifying Knowledge Base and Database Design*, Springer-Verlag.
- De Marco T. 1978 *Structured Analysis and Systems Specifications*, Yourdon Inc..
- Frost R. A. 1986 *Introduction to Knowledge Base Systems*, MacMillan.
- Gane C. and T. Sarson T. 1979 *Structured Systems Analysis*, Prentice-Hall.
- Gérard Genette G. 1983 *Narrative Discourse – an Essay in Method*, trans. Jane E. Lewin, New York
- Greimas A. J. 1966 *Sémantique structurale*, Paris.
- Gunder A. 1999 "Berättelsens spel. Berättarteknik och ergodicitet i Michael Joyces afternoon, a story", *Human IT 3/1999*. URL: <http://www.hb.se/bhs/ith/3-99/ag.htm>.
- Oliver J. H. 2001 "Polygon Destinies: The Production of Place in the Digital Role-Playing Game", *Computational Semiotics for Games and New Media (COSIGN)*, 10-13 September 2001.
- Lindley C. 2002 "The Gameplay Gestalt, Narrative, and Interactive Storytelling", *Computer Games and Digital Cultures Conference*, June 6-8, Tampere, Finland.
- Partridge A. 2002 *Real-Time Interactive 3D Games*, Sams.
- Rimmon-Kenan S. 1998 *Narrative Fiction: Contemporary Poetics* London and New York
- Yourdon E. and Constantine L. L. 1979 *Structured Design: Fundamentals of a Discipline of Computer Program and System Design*, Prentice-Hall.