

# Non-Linear Interactive Stories in Computer Games

*Olav Bangsø and Ole Guttorm Jensen*

*Game Approaches / Spil-veje. Papers from spilforskning.dk Conference, august 28.-29. 2003*  
Spilforskning.dk 2004. ISBN 87-990066-1-8

*The paper introduces non-linear interactive stories (NOLIST) as a means to generate varied and interesting stories for computer games automatically. We give a compact representation of a NOLIST based on the specification of atomic stories, and show how to build an object-oriented Bayesian network (OBN) from it, which is used to automatically generate stories based on past events in the story and the interaction of the player.*

## 1 Introduction

The narrative in many computer games (particular the adventure genre) has often been compared to movies that stop at certain points to present the viewer with a number of choices that determines how the narrative progresses from that point on. These kind of stories are highly linear with minimal opportunity for the player to interact.

Experiences with these games will in terms of narrative be virtually identical, resulting in much lower replayability than other genres of computer games, e.g. strategy games. From an industry perspective, the cost of producing a huge amount of material that is not activated during play prevents the development of stories with less linearity and more freedom for the player to interact. Ideally, game developers want to keep costs down while maximizing the replay value of a game, so players may experience different stories each time they play even with the same interaction.

As a step towards this goal, we propose *non-linear interactive stories* (NOLIST) for computer games. A NOLIST consists of a set of *atomic stories*, which are basically small story parts with some information regarding how they can fit into the full story, and a system for composing the full story from the atomic stories and the player's interaction. The general idea is that while the narrative unfolds, the system continuously considers the past, i.e. which atomic stories have been used in the story so far, and then generates a set of possible continuations of the story, i.e. sequences of atomic stories that lead to an end. Each continuation is then scored to determine how "good" in terms of narrative it is, and one of the highest scoring continuations is picked by the system or the player. Note that after picking a continuation we do not necessarily stick with it. At any time, the player or the system may pick another continuation or developments in the story may even make the system discard that continuation. Also note that only some of the possible continuations are generated at each point in the story. This means that even if two players interact in the same way, they are unlikely to experience the exact same story.

The paper proceeds as follows. In Section 2 we define a NOLIST and provide a compact representation for it. Section 3 gives an example of a NOLIST based on a detective story, and in Section 4 we show how to construct an object-oriented Bayesian network (OOBN) for a NOLIST. In Section 5 we show how to compose a story from an OOBN. Conclusions and future research is given in Section 6.

## 2 Non-Linear Interactive Stories (NOLIST)

A *non-linear story* contains several possible sequences of events and may have different start and end points. The sequences can have arbitrary interleaving of events and events from one sequence can influence the order of events in other sequences and their applicability, i.e. some events can exclude other events from happening within the context of the story. In a non-linear interactive story the person being told the story can influence the sequence of events. We will let an unconstrained NOLIST consist of a set of atomic story parts, which can be told in any order. If there are constraints of the kind *story part A must have happened before story part B* we will for ease of exposition use the term NOLIST. A NOLIST is inherently exponential in size, as after each part of the story is finished, almost any other part can be told. But by letting the story be build from small parts, only these and the constraints need to be specified, and not all possible orders in which the story parts can be told. More formally:

### Definition

*An atomic story, AS is a tuple (P,E,N), where P are the prerequisite events that must have occurred for AS to be told, E are the events triggered by telling AS, and N is the narrative. An end atomic story is an atomic story that has been marked as a meaningful last atomic story in a NOLIST.*

Note that the prerequisites can also be of the form that an event has not occurred. An atomic story where the prerequisites are met is called a legal atomic story. This way of dividing a NOLIST into small parts where there is no interaction or non-linearity allows the game designer to make a very compact representation of the NOLIST; only the events that must occur before a given atomic story is legal and the events triggered by that atomic story needs to be specified. If an atomic story is marked as an end atomic story, it is possible to end the NOLIST when this atomic story has been told. We furthermore allow a hierarchy of sets of atomic stories called chapters, sub-chapters, and so on.

### Definition

*A chapter C is a tuple (P,E,S) where P and E are prerequisite and events triggered, respectively. S is a set of sub-chapters and atomic stories. A sub-chapter is a chapter that is Defined as part of a chapter. An end chapter is a chapter marked as being an ending to the structure encapsulating it (chapter or NOLIST). End atomic stories will only be endings to the encapsulating structure.*

The definition of chapters and sub-chapters are very close to that of atomic stories, so if you know how to specify an atomic story, it is easy to also incorporate a chapter structure on top of them. A legal chapter is one where the prerequisites are met. Note that atomic stories and sub-chapters can occur in different chapters.

### 3 A NOLIST Example

In the following, an example of a NOLIST is given. The example is a detective story where the player takes the role of the investigator with the goal of finding out who killed the victim by looking at the crime scene and interviewing the suspects *A, B, C,* and *D*. The story is divided into 6 chapters, a starting chapter where the crime scene is investigated, four chapters where different suspects are interviewed, and an ending where the murderer is revealed. Note that the first 5 chapters can be told several times as long as the crime scene is the first chapter and the revealing of the murderer is the last. We will only fill in the prerequisites and events triggered, as the story part is better left to an author of detective stories. All atomic stories are marked as possible endings for the chapter in which they are defined.

#### 3.1 Investigating the Crime Scene

For convenience this chapter will be called *I* and the elements in the tuple will be subscripted with an *I*. The chapter has are no prerequisites, so  $P_I$  is empty. The events,  $E_I$ , records any uncovered evidence regarding the motive for the murder,  $e_m$ , with three states; *Y* (motive *Y*), *X* (motive *X*) and ``not found" (no motive determined), and evidence regarding the opportunity to commit the murder,  $e_o$ , also with three states; *M* (opportunity *M*), *N* (opportunity *N*), and ``not found" (no opportunity determined). Each suspect also have events to monitor whether they in fact had opportunity and motive to commit the murder e.g.  $Opp_A$  and  $Mot_A$  with the states *U* (unknown), *N* (suspect had no motive/opportunity), *Y* (suspect had motive/opportunity), *C* (suspect had motive/opportunity and it has been confirmed by a second witness or clue).

The atomic stories of this chapter are:

Subchapter	Prerequisites	Events Triggered	Narrative
1.1	$e_m=U$	$e_m=X$	Examine scene 1
1.2	$e_m=U$	$e_m=Y$	Examine scene 2
1.3	$e_o=U$	$e_o=N$	Examine victim 1
1.4	$e_o=U$	$e_o=M$	Examine victim 2
1.5	$V=NF$	$Opp_A=Y, V=F$	A found victim
1.6	$V=NF$	$Opp_B=Y, V=F$	B found victim
1.7	$V=NF$	$Opp_C=Y, V=F$	C found victim
1.8	$V=NF$	$Opp_D=Y, V=F$	D found victim

Note that event *V* is not triggered by the chapter but by the atomic stories in the chapter, and is used to determine if the knowledge on who found the victim has been revealed or not. This event is local to the chapter, because it will not be used by other

chapters. The states are *F* (found) and *NF* (not found). The numbers are there for ease of reference.

### 3.2 Interviewing A

This chapter can trigger the motive and opportunity events for the suspects except the motive for *A*, *A* will not incriminate himself. The prerequisites for the chapter are the disjunction of the prerequisites of all the atomic stories.

Subchapter	Prerequisites	Events Triggered	Narrative
2.1	<i>Mot B=U</i>	<i>Mot B=Y</i>	Ask about B 1
2.2	<i>Opp B=U</i>	<i>Opp B=Y</i>	Ask about B 2
2.3	<i>Opp B≠U</i>	<i>Opp B=N, Opp A=N</i>	Ask about B 3
2.4	<i>Mot C=Y, e m=X</i>	<i>Mot C=C</i>	Ask about C 1
2.5	<i>Opp C=U</i>	<i>Opp C=Y</i>	Ask about C 2
2.6	<i>e o=X</i>	<i>Opp C=C</i>	Ask about C 3
2.7	<i>Mot D=U</i>	<i>Mot D=Y</i>	Ask about D 1
2.8	<i>Opp D=U</i>	<i>Opp D=Y</i>	Ask about D 2
2.9	<i>Opp D≠U</i>	<i>Opp D=N, Opp A=N</i>	Ask about D 3

### 3.3 Interviewing B

This chapter can trigger the motive and opportunity events for the suspects except the motive for *B*, *B* will not incriminate himself. The prerequisites for the chapter are a disjunction of the prerequisites of all the atomic stories.

Subchapter	Prerequisites	Events Triggered	Narrative
3.1	<i>Mot A=U</i>	<i>Mot A=Y</i>	Ask about A 1
3.2	<i>Opp A=Y, e o=N</i>	<i>Opp A=C</i>	Ask about A 2
3.3	<i>Opp B=Y, e o=M, Mot B=U</i>	<i>Opp B=C</i>	Ask to confess
3.4	<i>Mot C=U</i>	<i>Mot C=Y</i>	Ask about C 1
3.5	<i>Opp C=Y, e o=N</i>	<i>Opp C=C</i>	Ask about C 2
3.6	<i>Opp C≠U</i>	<i>Opp C=N, Opp B=N</i>	Ask about C 3
3.7	<i>Mot D=Y</i>	<i>Mot D=C</i>	Ask about D 1
3.8	<i>Opp D=U</i>	<i>Opp D=Y</i>	Ask about D 2
3.9	<i>e o≠N</i>	<i>Opp D=N, Opp A=Y</i>	Ask about D 3

### 3.4 Interviewing C

This chapter can trigger the motive and opportunity events for the suspects except the motive for *C*, *C* will not incriminate himself. Furthermore *e m* may be triggered. The prerequisites for the chapter are a disjunction of the prerequisites of all the atomic stories.

Subchapter	Prerequisites	Events Triggered	Narrative
------------	---------------	------------------	-----------

4.1	$e_m \neq X$	$e_m = Y$	Ask about A 1
4.2	$Mot_A = Y, e_m = X$	$Mot_A = C$	Ask about A 2
4.3	$Opp_A = U$	$Opp_A = Y$	Ask about A 3
4.4	$Mot_B = U$	$Mot_B = Y$	Ask about B 1
4.5	$Opp_B = Y$	$Opp_B = C$	Ask about B 2
4.6	$e_o \neq M$	$Opp_C = N, Opp_D = N$	Ask about alibi
4.7	$Mot_D = Y$	$Mot_D = C$	Ask about D 1
4.8	$Opp_D = Y, e_o = M$	$Opp_D = C$	Ask about D 2

### 3.5 Interviewing D

This chapter can trigger the motive and opportunity events for the suspects except the opportunity for *D* and *B*. Furthermore  $e_o$  may be triggered. The prerequisites for the chapter are a disjunction of the prerequisites of all the atomic stories.

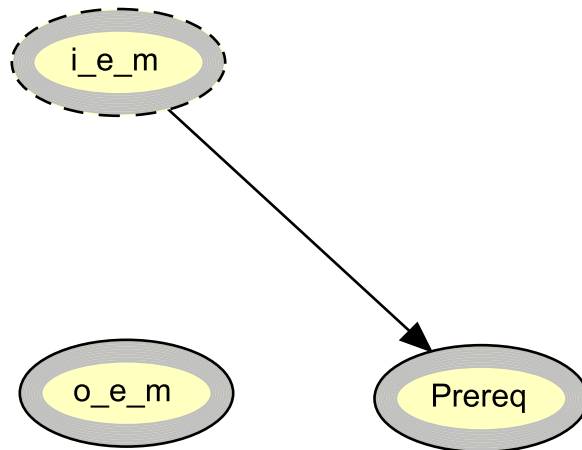
Subchapter	Prerequisites	Events Triggered	Narrative
5.1	$Mot_A = U, Mot_C \neq C$	$Mot_A = N, Mot_C = Y$	Ask about A 1
5.2	$e_m = X$	$Mot_A = C$	Ask about A 2
5.3	$Opp_A = U$	$Opp_A = Y$	Ask about A 3
5.4	$Mot_B = Y, e_m = Y$	$Mot_B = C$	Ask about B 1
5.5	$e_o \neq M$	$e_o = N$	Ask about B 2
5.6	$Mot_C = U$	$Mot_C = Y$	Ask about C 1
5.7	$Opp_C = Y, e_o = N$	$Opp_C = C$	Ask about C 2
5.8	$e_m = Y$	$Mot_D = C$	Ask to confess

### 3.6 Revealing the murderer

To reveal the murderer both motive and opportunity must be confirmed for one of the suspects, which constitutes the prerequisites for the chapter. The chapter is also marked as an end chapter indicating that revealing the murderer will end the story. There are four atomic stories in this chapter, one for each suspect with the prerequisites that both motive and opportunity are confirmed for the suspect in question.

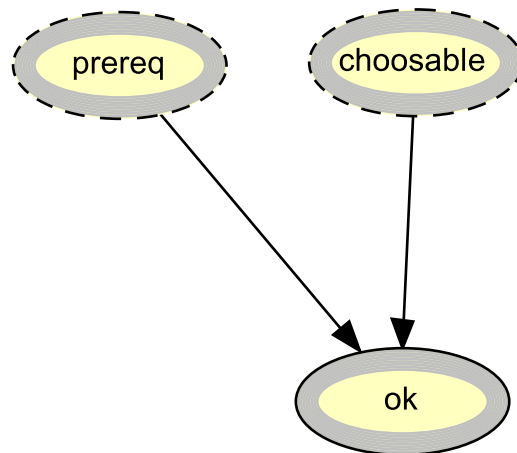
## 4 Creating an OOBN for a NOLIST

By viewing the events as variables, a NOLIST is close to being an OOBN. Each atomic story and each chapter will be a class. A class for an atomic story will consist of the prerequisites as input nodes and the events triggered as output nodes. A class for "Examine scene 1" in the investigate crime scene chapter can be seen in Figure 4.1. We need to trigger another event stating whether the prerequisites are fulfilled or not. As we will show later, this will be used by the chapter class to determine if the atomic story can be told or not, so this must be an output of the atomic story.



**Figure 4.1:** A class for the atomic story "Examine scene 1". The special event output node *Prereq* has two states, *Y* and *N*, to indicate whether the prerequisites are satisfied or not. *i\_e\_m* is the input prerequisite and *o\_e\_m* is an output node.

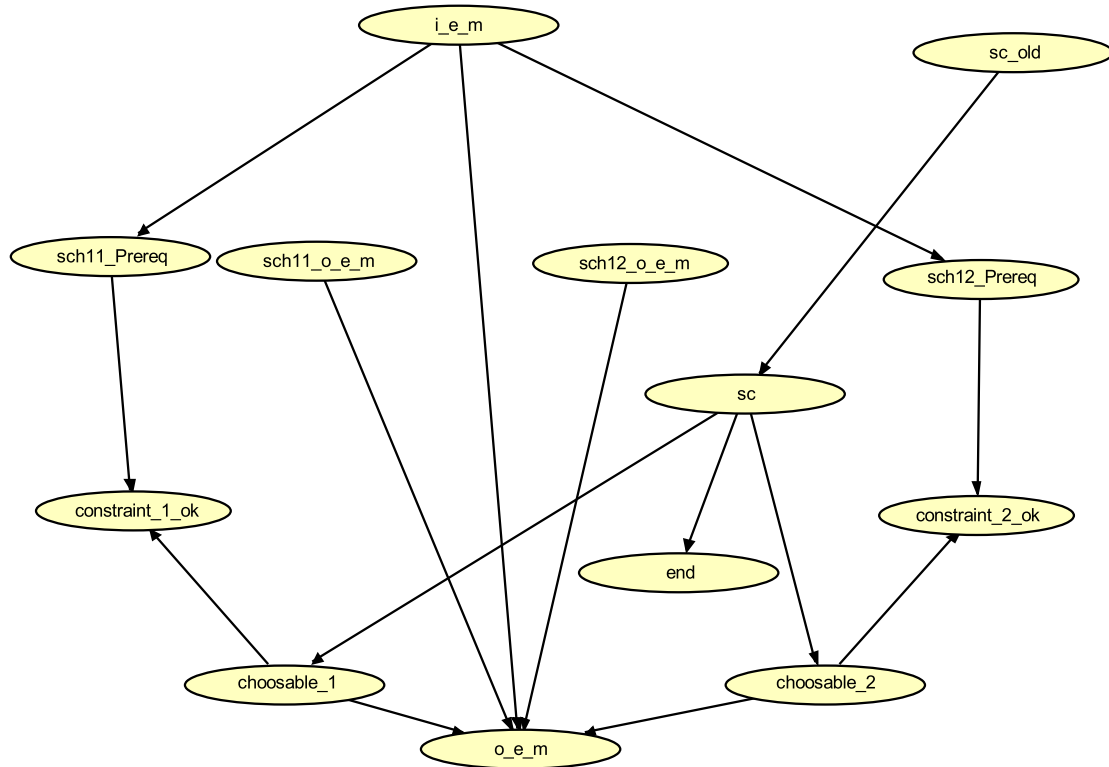
The chapters are a bit more difficult, as it must be possible to tell any legal part and also to keep telling parts until an end part has been told, and even then parts can still be told, as long as the last part told in a chapter is an end part. To handle this we make a representation of a chapter that will tell one part and make a time-slice instance of it to tell several of the parts.



**Figure 4.2:** The constraint class instantiated as a time-slice with one copy for each part of the chapter.

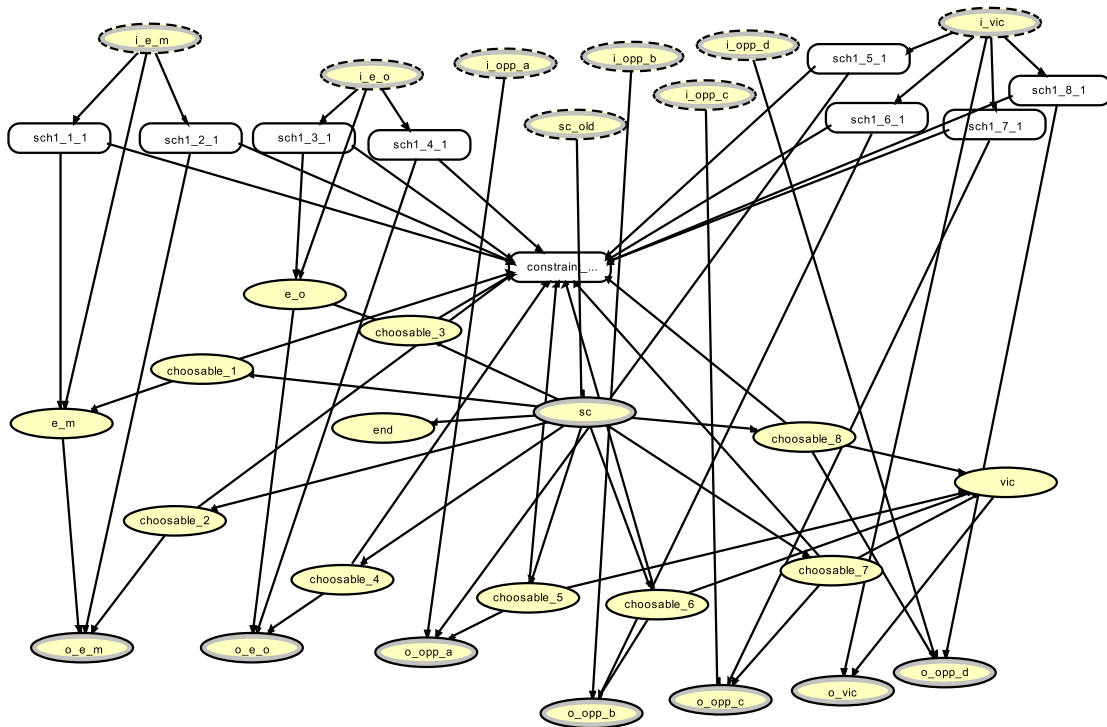
A chapter class has to make sure that only one part of the chapter is told, and that only legal parts can be told. This is done by using an instance of the class for each part, a time-slice instance of the constraint class in Figure 4.2 with one time-slice for each part of the chapter. Evidence will be entered in the *Ok* nodes to make sure that only legal parts can be chosen. To ensure that at most one part is told a variable with a state for each part and one for choosing nothing is created, and a variable for each part monitoring if that part is choosable is created as a child of this. *sc* is the variable describing legal parts, the *choosable* variables are true, if the corresponding part is legal. Note that *sc* has *sc\_old* as a parent, this is to ensure that if nothing is told in a chapter instance, nothing will be told in subsequent chapter instances, in the chapter in question. The default potential for *sc\_old* will be uniform over its states excluding nothing. Also, nothing can only be told if the last part told was an end part. Making sure that a chapter ends with an end part is done by introducing the variable *end* with

two states,  $Y$  and  $N$  as a child of  $sc$  and letting the CPT be constructed so  $end$  is in state  $Y$  if the part told in  $sc$  is an end part and  $N$  otherwise. Note that if  $sc$  is in the state  $nothing$ , then the last part actually told was an end part, so  $nothing$  is also an end. In Figure 4.3 the underlying network for an example with two parts, and one event,  $e_m$ , is shown, evidence will be entered in the two  $ok$  variables to ensure that the *choosable* variables are only  $Y$  if the prerequisites for the corresponding part are met.



**Figure 4.3:** The part of a chapter class ensuring that at most one part is told, and that only legal parts are told.

All that is left for the chapter classes is to make sure that the appropriate variables are available for the parts, and that the outputs are updated according to the part told. Which is done by having the union of inputs of the parts as input to the chapter class, and the union of the outputs of these as output. In Figure 4.4 the chapter class for the first chapter of the detective story can be seen. Each output have the corresponding input, if applicable, as a parent to make sure that if nothing is told nothing happens, to the events states. Furthermore each output have the *choosable* variables corresponding to the parts where they may be changed as parents, and the appropriate output of those parts are also parents. We have used parent divorcing to simplify the CPTs, but as an example look at  $o_e_m$ , the output of the chapter class for the event  $e_m$ . It can be changed in parts 1.1 and 1.2 ( $sch1_1$  and  $sch1_2$ ), so  $choosable_1$  and  $choosable_2$  are parents along with the  $e_m$  outputs of these instances. The CPT of  $o_e_m$  is the same as  $i_e_m$  if none of the *choosable* variables are true, otherwise it will be the same as the  $e_m$  variable of the instance corresponding to the one that is. The structure and the evidence on the  $ok$  variables ensures that at most one *choosable* variable can be true. Note that the chapter class should be a time-slice class where each output self-referenced by the corresponding inputs, if it exists, this hasn't been done as HUGIN cannot handle self-references, but it can be modeled.



**Figure 4.4:** The complete chapter class for the example. The instances have not been expanded to keep an overview.

Once all chapter classes are completed, the overall story class can be made. The story class will be constructed in the same way as the chapter classes, having the chapters as parts. An OOBN for the example can be generated automatically from the specification in Section 3. We will call an OOBN generated in this way a NOOBN.

## 5 Composing a Story from an NOOBN

Once the NOOBN is finished, we need to generate a story from it, this will be done by sampling. Once a set of stories have been generated we need to tell one that is interesting. Furthermore we need to incorporate interactivity. The algorithm for telling stories from an NOOBN is as follows:

- 1) Insert evidence.
- 2) Sample all *sc* variables.
- 3) Choose next part to be told.
- 4) Repeat until the story is at an end.

The evidence entered will be on the *ok* variables of *constraint* instances, on the *end* variables of the last time-slice of each part and of the overall story, and on the *sc* variables of the parts already told.

We have chosen to sample stories, instead of using e.g. maxprop. This is to ensure that we get different stories, even if everything so far has been the same. Sampling is done by starting with the first part and hence the first *sc* variable that hasn't received evidence in this part yet. There are different ways this sampling can be done, one is to

incorporate the probability distribution in the variable by weighting the possible states with their probability. However we are not really interested in getting highly likely stories, we want stories that are different, so we are only interested to see if they are possible or not. We propose to choose at random one of the possible states. We will then sample the next *sc* variable given the previous choice. This however requires a propagation of the network with all the evidence from step 1. and evidence on the previous sampled *sc* nodes as we have evidence downstream. We will continue to sample *sc* variables as long as we have not reached the end of the part we are sampling. We have reached the end of the part if there are no more unsampled *sc* nodes, or we have chosen nothing in one, as this will entail all the rest of the *sc* nodes to be nothing as well. When sampling in the first part is done, we continue with the second part, and so on. All the *sc* nodes will now be a complete story, but we may have sampled an uninteresting story. To avoid telling uninteresting stories, we will generate a number of stories, so we can, hopefully, choose one that is interesting.

To choose an interesting story, we need a way to score a story. The easy way is to let every atomic story have an interestingness score, and let the story's score be the sum of those. This approach will be easy to specify and to compute, but it will not capture all the intricacies of a well told story, e.g. in a detective story it usually good form to let the evidence point strongly at one of the subjects, only to reveal that this subject could not have done the crime. We have yet to come up with a good way to score stories, especially as the scoring rules are different from genre to genre. But assuming that we have a way to score stories, we can then pick one of the generated stories at random using their score as a weight. We have chosen a random choice to get even more variation in the stories, but choosing the best is also viable. Once a story is chosen we tell the first atomic story, and add it to the list of *sc* that have been told, and start the process again until the story ends.

There are several ways to incorporate interactivity in this scheme, and it is up to each application to choose the one that best fits the story. One is to let the user make all the choices in step 3. Step 2 will then serve as a way to find which choices will lead to a possible ending and the user is limited to those choices. In the other extreme it is possible to let only let the user choose to be told a story or not. mixtures of these are possible, e.g. let the user choose the chapters, but not the stories, or the other way around is also possible.

## 6 Conclusions and Future Research

In this paper, we introduce non-linear interactive stories (NOLIST), as a means to generate varied and interesting stories for computer games automatically based on a small set of specified atomic stories. We provide a compact representation for a NOLIST and show how to create an object-oriented Bayesian network (OOBN) for it. We give an algorithm for composing a story from an OOBN that uses sampling to achieve different but consistent stories.

The work presented in this paper is still preliminary and subject to ongoing research. We are currently implementing a prototype system in MatLab to provide a proof of concept and to investigate and evaluate different sampling techniques for story generation. A direction for future research is to investigate how to quantify the interestingness of a story. This is essential for allowing a high degree of interactivity

while keeping an interesting narrative in a game, because game designers cannot realistically take into account all the possible ways in which players may interact. Another avenue of research, is to investigate the usefulness of atomic stories as building blocks for a NOLIST. This would require hands-on experience, and be the basis for the development of tools assisting game designers in specifying NOLISTs.