NON-LINEAR INTERACTIVE STORYTELLING USING OBJECT-ORIENTED BAYESIAN NETWORKS

Olav Bangsø[1], Ole G. Jensen[1], Finn V. Jensen[1], Peter B. Andersen[2], and Tomas Kocka[3]

[1]Department of Computer Science, Aalborg University,
Frederik Bajers Vej 7E, DK–9220 Aalborg Øst, Denmark,
E-mail: <bangsy | guttorm | fvj>@cs.aau.dk

[2]Department of Information and Media Studies, University of Aarhus,
Helsingforsgade 14, DK–8200 Aarhus N, Denmark,
E-mail: pba@imv.au.dk

[3]ADASTRA, s.r.o.,
Beneovská 10, 101 00 Praha 10, Czech Republic,
E-mail: tomas.kocka@adastracorp.com

KEYWORDS

Interactive narrative, interactive drama, object-oriented Bayesian networks.

ABSTRACT

Narration and interaction are often viewed as contrary properties in computer games. Games with a high degree of interaction fail to provide a coherent narration and the player's interaction seldom has any direct impact on the narrative. Games with a high degree of narration often tells a linear story similar to books or movies with little room for the player to interact. We propose *non-linear interactive storytelling* (NOLIST) as a first step towards developing games with a high degree of interaction and a coherent narrative. The main idea is that the narrative is not fixed from the beginning but instead constructed as the game progresses based on the player's interaction. We provide a simple model that allows writers to specify a NOLIST as a set of *actions* which the game engine then combines to create the narrative. Finally, we propose to develop a game engine using Bayesian networks to model the probability of the possible narratives that can be created from the actions, and use this knowledge to create better narratives.

INTRODUCTION

Narration and interaction are often considered as two incompatible properties in computer games. This has lead to the distinction between games of progression and games of emergence [Juul, 2004]. Games of progression have been compared to movies that stop at certain points to allow the player to choose among a set of options which determine how the narrative progresses from that point on. In these games, narration is highly linear with limited player interaction, so playing the game more than once rarely provides the player with a new gaming experience. Games of emergence define a set of simple and deliberate rules which when combined emerge into more complex patterns, and thus motivate the player to develop more advanced strategies for playing the game. The high level of player interaction is achieved at the expense of a coherent narrative. Players rarely experience events later in the game as direct consequences of earlier events, and much of the interaction has no impact on the narrative [Mallon and Webb, 2000].

This paper proposes *non-linear interactive storytelling* (NOLIST) as a first step towards the development of games with both a high degree of player interaction and a coherent narrative. The main idea is that the narrative is not fixed from the outset, but instead constructed as the game progresses. The outcome of events that occur in the game whether caused by the player's direct interaction or by agents in the game world change the likelihood of past events (not observed by the player) having occurred and the probability that certain future events can occur. At any point in the game, the narrative consists of the events observed by the player. These events determine the probability and possibility of different pasts and futures for the narrative. The player's interaction is restricted to events that are consistent with the possible pasts and futures in order to ensure a coherent narrative. As more events are observed by the player, the set of possible pasts and futures narrows. Consequently, the player's choices become more and more restricted as the game progresses until all the events of the narrative are determined.

We propose a framework to develop NOLIST in computer games. A NOLIST game engine specifies a set of actions which are the building blocks of the narrative. An

action could, e.g., be a small movie clip, a sentence in a dialog, or the description of an event. The game engine maintains a model of the possible pasts and the possible futures for the narrative. The game is played in rounds, where in each round the game engine first determines which actions are possible. These actions must be part of a possible future and be consistent with a possible past. The player then chooses one of them. The chosen action is played out in the game and then part of the narrative. The chosen action influences the possible pasts and futures for the narrative. E.g., in a murder story the investigator (the player) finds a smoking gun close to the spot where someone was killed. This action influences the past by making it more probable that the gun was used to kill the victim and less probable that the victim was stabbed to death. In turn, all suspects with access to the gun are more likely to be the murderer. Changes to the possible pasts influence the possible futures. E.g., the suspects with access to the gun are now more likely to try to conceal their actions and mislead the investigator. The action also influences the possible futures directly. E.g., it is now more probable that a bullet will be found in the victim's body if it is examined. When the game engine has determined how the chosen action influences the possible pasts and futures, a new round can commence. Some actions are possible endings. The game can only end immediately after the game engine has played out such an action, however, this is not mandatory.

NOLIST is related to interactive drama. The narrative engine, IDtension, calculates the set of all possible actions of the characters based on the current state in the world of the story and ranks them according to a user model for their narrative effects [Szilas, 2003]. The NOLIST game engine ensures consistency between past events observed by the user and future events as well as prevents stories with no endings. The narrative quality of stories is not considered in this paper, however, with NOLIST it is possible to evaluate the narrative quality based on the probability of the possible futures of each action. In Facade interactive dramas are divided into beats and semi-autonomous agents with a drama manager are used to choose among the possible beats [McKee, 1997, Mateas and Stern, 2003]. Each beat has a tension value and beats are chosen to best fit the Aristotelian story tension value arc. It is the responsibility of the author to ensure that all states have possible beats. In NOLIST atomic actions resemble beats, and the engine ensures that the story never reach a state where no actions are possible.

NON-LINEAR INTERACTIVE STORYTELLING (NOLIST)

A story is usually divided into a number of smaller parts, we use the term chapter, but any subdivision is valid and supported. The chapters are supposed to be read in sequence, so the first chapter begins the story while the last chapter ends it. Before reading each chapter, the reader is required to know about certain characters and events in the story (those described in all chapters preceding this chapter). Reading the chapters in any other order often makes the story confusing and incoherent. We say that such a story is linear because all the chapters must be read in a specific order. In contrast, a non-linear story allows chapters to be read in different orders and not all chapters have to be read. We cannot expect a coherent narrative to result from any random order of chapters, so each chapter has some prerequisites that must be satisfied by the preceding chapters. E.g., to identify the murderer, a motive and an opportunity must have been established. However, the details of what the motive and opportunity are or how they were established are not relevant and can be established by different sequences of preceding chapters. Therefore, with much fewer chapters a non-linear story can represent many different linear stories. In a non-linear interactive story the reader can influence the order of chapters.

We introduce actions as the building blocks of a non-linear interactive story. Some actions are possible endings (denoted by a suffix '*'). An *action* is characterized by its content, prerequisites, and effects. The *content* encapsulates a set of actions, analogical to a chapter being divided into sub-chapters. If the content is empty, then the action is *atomic*. The *prerequisites* are the events that must have occurred before the action can be performed and the *effects* are the events that occur as a result of performing the action. An *event* is a simple statement such as *the gun is the murder weapon* or *if John has a motive it is not known*. The content of an action may encapsulate actions which in turn encapsulate other actions. This leads to an *action hierarchy* such as the one illustrated in Figure 1. In the figure the action *Examine the crime scene* encapsulates the two actions X *found at crime scene* and Y *found at crime scene*, both of which are atomic actions (since their content is empty) and possible endings (denoted by the '*').

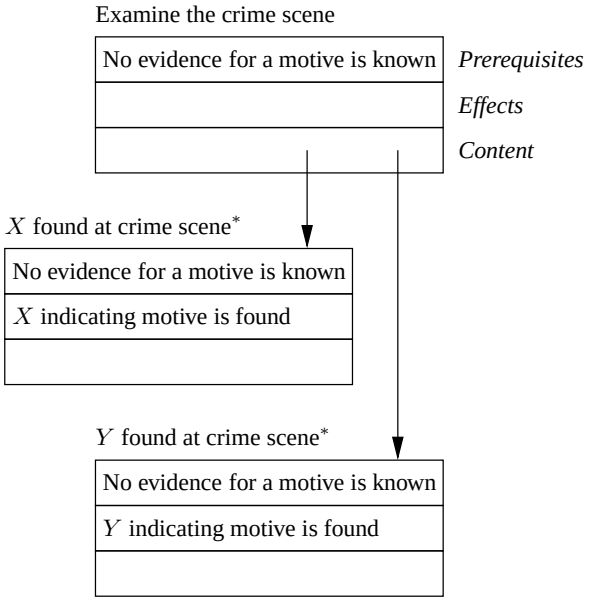

Figure 1: Action hierarchy for *Examine the crime scene*.

A *non-linear interactive story* consists of an action and a past. The *past* is a set of events that is believed to hold at a particular point in the narrative, and only actions whose prerequisites are satisfied by the past can be performed. Whenever an action is performed, the past is changed to reflect the effects of that action.

An Example of a Non-Linear Interactive Story

To help illustrate non-linear interactive stories and actions we provide an example. The example is a murder story where the player takes the role of an investigator trying to solve a murder case by investigating the crime scene and interviewing the three suspects named A, B, and C. Figure 2 depicts the action hierarchy for the murder story. At the top level (1) is the Murder story action with a content of five actions: Investigate the crime scene (1.1), Interview A, B, and C (1.2-1.4), and Reveal the murderer (1.5). Actions at the lowest level are atomic.

The prerequisites and effects of the atomic actions are in the appendix. The effects of all non-atomic actions in the murder story are empty, and the prerequisites of any non-atomic action is the disjunction of the prerequisites of the actions in its content.

When the game begins, the past is initialized as follows:

No evidence for a motive is known
No evidence for an opportunity is known
It is not known who found the victim
It is not known if A had a motive
It is not known if B had a motive
It is not known if C had a motive
It is not known if A had an opportunity
It is not known if B had an opportunity
It is not known if C had an opportunity

In order to identify the murderer, the investigator must establish a strong motive and a clear opportunity for at least one of the suspects. Note that at this point, the game has not decided who actually murdered the victim. This will be determined as the game progresses based on the player's interaction with the game.

A Game Example using the Murder NOLIST

In this section we describe an example gaming session using the murder story described in the previous section. When using a NOLIST for a game, some actions are chosen by the player or others by the game engine. In this example, all non-atomic actions are chosen by the player while the game engine chooses among the atomic actions. Table 1 summarizes the gaming session.

The game begins at the top level of the action hierarchy, where only the Murder story action is available. To perform this action, we must perform actions in its content whose prerequisites are satisfied by the (initial) past until a possible ending is performed. All actions at this level of the action hierarchy are available except 1.5, and the player chooses 1.1. Again, the player must choose among the actions in the content of action 1.1, and chooses 1.1.1.

1	Murder story*
1.1	Investigate the crime scene
1.1.1	Examine the victim*
1.1.1.1	M found on victim*
1.1.1.2	N found on victim*
1.1.2	Examine the crime scene*
1.1.2.1	X found at crime scene*
1.1.2.2	Y found at crime scene*
1.1.3	Ask who found the victim*
1.1.3.1	A found the victim*
1.1.3.2	B found the victim*
1.1.3.3	C found the victim*
1.2	Interview A
1.2.1	Ask A about B^*
1.2.1.1	A indicates weak motive for B^*
1.2.1.2	A indicates vague opportunity for B^*
1.2.1.3	A and B have an alibi*
1.2.2	Ask A about C^*
1.2.2.1	A indicates strong motive for C^*
1.2.2.2	A indicates vague opportunity for C^*
1.3	Interview B
1.3.1	Ask B about A^*
1.3.1.1	B indicates weak motive for A^*
1.3.1.2	B indicates clear opportunity for A^*
1.3.2	Present evidence to B^*
1.3.2.1	B acknowledges clear opportunity*
1.3.3	Ask B about C^*
1.3.3.1	B indicates weak motive for C^*
1.3.3.2	B indicates clear opportunity for C^*
1.3.3.3	B and C have an alibi*
1.4	Interview C
1.4.1	Ask C about A^*
1.4.1.1	C provides evidence indicating motive*
1.4.1.2	C indicates strong motive for A^*
1.4.1.3	C indicates vague opportunity for A^*
1.4.2	Ask C about B^*
1.4.2.1	C indicates weak motive for B^*
1.4.2.2	C indicates clear opportunity for B^*
1.5	Reveal the murderer*
1.5.1	A is the murderer*
1.5.2	B is the murderer*
1.5.3	C is the murderer*

Figure 2: Action hierarchy for the murder story. Actions marked with * are possible endings.

P: 1	Murder story
P: 1.1	Investigate the crime scene
P: 1.1.1	Examine the victim
G: 1.1.1.2	N found on victim
P: 1.1.2	Examine the crime scene
G: 1.1.2.1	X found at crime scene
P: 1.1.3	Ask who found the victim
G: 1.1.3.1	A found the victim
P: 1.2	Interview A
P: 1.2.2	Ask A about C
G: 1.2.2.2	A indicates vague opportunity for C
G: 1.2.2.1	A indicates strong motive for C
P: 1.3	Interview B
P: 1.3.3	Ask B about C
G: 1.3.3.3	B and C have an alibi
P: 1.3.1	Ask B about A
G: 1.3.1.1	B indicates weak motive for A
G: 1.3.1.2	B indicates clear opportunity for A
P: 1.4	Interview C
P: 1.4.1	Ask C about A
G: 1.4.1.2	C indicates strong motive for A
P: 1.5	Reveal the murderer
G: 1.5.1	A is the murderer

Table 1: An example gaming session of the murder story.

The contents of action 1.1.1 contains two atomic actions both with satisfied prerequisites. The game engine chooses among atomic actions and performs 1.1.1.2. The action is performed immediately (since it is atomic) and its effects update our belief in which events occurred in the past by replacing the statement *No evidence for an opportunity is known* with N *indicating opportunity is found*. Since 1.1.1.2 is a possible ending, the enclosed action (1.1.1) can be completed. Action 1.1.1 is also a possible ending, so the player may choose to complete action 1.1 as well. However, the player chooses to complete the two remaining actions in its content instead before completing action 1.1. At this point, the player chooses to interview A and establish a vague opportunity for C to commit the murder and a strong motive. The player now questions B about C hoping to establish a clear opportunity for C and thus reveal C as the murderer (action 1.3.3). The game engine has two atomic actions with satisfied prerequisites to choose from: 1.3.3.2 and 1.3.3.3. Either a clear opportunity is established for C or B provides an alibi for both B and C. In the example, the game engine selects the latter atomic action. Consequently, neither B nor C are likely suspects, and the player starts to question them about A (who apparently tried to frame C). This line of questioning leads to a strong motive and a clear opportunity for A, and A is finally revealed as the murderer by action 1.5.1. This action is a possible ending and action 1.5 can now be completed. Action 1.5 is also a possible ending, so the top level action can finally be completed, and the game ends.

In the example we did not specify *how* the game engine chooses which atomic story part to play next. Satisfaction of the prerequisites of an atomic story part is not sufficient to ensure a coherent and interesting story. E.g., in the murder story it is entirely possible for more than one suspect to be the murderer or for all suspects to have an alibi. In the latter case, the prerequisites for the chapter marked as an ending will never be satisfied, so the story continues indefinitely. In the former case, only one of the suspects will be revealed before the story ends. To avoid such inconsistent stories, the game engine has to consider the possible futures for each of the atomic story parts to be played next. Only atomic story parts with acceptable futures and satisfied prerequisites can be played next. E.g., in the murder story, atomic story parts leading to everyone having an alibi or more than one suspect having a clear opportunity and a strong motive will never be played. As a consequence some actions may become unavailable to the player although they are associated with atomic story parts with satisfied prerequisites.

By evaluating the stories produced by the possible futures for an atomic story part, the game engine can avoid the least interesting stories. E.g., in round six of the murder story the game engine could have selected the atomic story part with the narrative: B indicates clear opportunity for C if evidence is known. This leads to C being the murderer. However, since all evidence points towards C from the beginning, the story is a trivial detective story. Clearly, what constitutes an interesting story varies a lot between genres. However, we believe that writers often have a fairly clear idea about how their own story should develop and that this can be modeled, at least in part, by considering how the past develops.

In the example we have only considered which atomic story parts are possible after each round of the game. In general, we would also like to know how probable they are. A story where all atomic story parts played are highly improbable might be fun to read but can easily become confusing and incoherent. Similarly, playing only the most probable atomic story parts produces a predicable and most likely boring story. Knowing the probability of each possible future helps the game engine to progress the story according to the stated intentions of the writer.

In the next section we introduce Bayesian networks for as they seem fit for modeling a NOLIST game engine that considers the probability of possible futures and avoids the pitfalls of stories without endings.

BAYESIAN NETWORKS

A Bayesian network is a graphical structure, which is used to represent cause-effect relations in a domain ([Pearl, 1988] and [Jensen, 2001]). In particular, they are widely used in domains with uncertainty attached to the impact of a cause. For example, if a investigator asks a suspect where he was at the time of the crime, the fact whether the suspect is guilty of the crime has a causal impact on the answer. However, even if the suspect was not at the crime scene at the time of the crime (and not guilty), you cannot be sure that he will tell the truth.

A Bayesian network consists of a structural part and a quantitative part. The structural part is a directed acyclic

graph (DAG), where nodes represent particular events, and the directed links represent cause-effect relations. A node has a finite set of states representing possibilities for this event. In Figure 3 the situation above is represented.

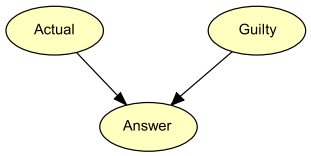

Figure 3: A DAG representing the situation where a investigator asks a suspect of his whereabouts at the time of the crime.

The node *Actual* represents the suspect's possible whereabouts, and it may have the states *home, crime scene, mistress,* and *work*. The node *Guilty* has the states *yes* and *no*, and the node *Answer* has the same states as *Actual* plus the state *no answer*. We refer to the relations in a DAG using family terms. For example, *Actual* is a parent of *Answer*, and *Answer* is a child of *Guilty*.

The strength of the cause-effect relations is represented through conditional probabilities. For each node you have to specify a probability distribution for its states given all possible configurations of its parent nodes. For the model in Figure 3 you for example specify the probability distribution for *Answer* given *Actual= home* and *Guilty=no*. This could be

$$P(Answer|Actual = home, Guilty = no) = (0.1, 0.1, 0.2, 0.3, 0.3)$$

You have to specify eight distributions of that kind. Furthermore, you also have to specify (prior) distributions for *Actual* and *Guilty*.

When Bayesian networks are used they are more complicated than the example in Figure 3. Figure 6 shows a slightly more complex model.

Basically, Bayesian networks are used to determine new probabilities given evidence. For example, when you know the answer of the question it is inserted in the network and used to determine the posterior probabilities for *Actual* and *Guilty*.

Object-Oriented Bayesian Networks

We utilize an extension to Bayesian networks called Object Oriented Bayesian Networks (OOBN) [Koller and Pfeffer, 1997, Bangsø and Wuillemin, 2000]. In the object oriented paradigm the basic component is an object; an entity with identity, state and behavior. Objects are grouped into classes. A class which is a description of a set of objects with the same structure, behavior and attributes. Whenever an object of a class is needed, an instance of that class is created. Note that each instance has a unique encapsulating class, the class in which they are instantiated.

A class is a Bayesian network fragment containing three sets of nodes:

- O: the set of output nodes; nodes that can be parents of nodes outside instances of the class.
- I: the set of input nodes; represents nodes that are not in the class, but are used as parents of nodes inside instances of the class. Input nodes cannot have parents in the class.
- P: the set of protected nodes; nodes that can only have parents and children inside the class.

The input and output nodes constitute the *interface*; the part of instances of a class that interfaces with the surroundings of the instance.

When an instance of a class is created, it can be linked to the rest of the network through the interface. To be able to do this linking a new type of link needs to be defined; the reference link. The child node in a reference link must be an input node and the parent is the node which is used as parent of the children of the input node, the parent and child must have the same number of states.

To allow the presence of input nodes without a parent, a default potential is introduced; a probability distribution over the states of the input node, used to create a regular node if the input node is not a child in a reference link. It is worth noting that the interface nodes are part of both the instance where they are defined and the class encapsulating that instance. This means that links from output nodes of an instance to nodes not in that instance (be it nodes in the encapsulating class or input nodes of other instances in the encapsulating class) are part of the specification for the encapsulating class.

By construction, instances of classes are inside a unique encapsulating class, ensuring that a tree of instances can be constructed. We will call such a tree an Instance Tree (IT). This tree will have the encapsulating class as the root, and each instance inside this class will define a sub tree with that instance as the root, and so on.

CREATING AN OOBN FOR A NOLIST

The specification of a NOLIST can be used to make a translation of the story into an OOBN. This OOBN will be used to choose actions for the game engine. The translation can be done automatically by letting :

- Prerequisites and effects for actions be variables.
- Actions be classes where the prerequisites are input nodes and the effects are output nodes.
- The action hierarchy be reflected in the IT.

In the following the translation will be outlined, and some problems and their solution described.

Atomic Actions

A class for an atomic action will consist of the prerequisites as input nodes and the effects as output nodes.

A class for one of the atomic actions under the "Examine the crime scene" action can be seen in Figure 4. *e_m* is the event modeling what evidence on motive has been found, it has the states *no evidence, M*, and *N*. This event is both a prerequisite and an effect, so it is present as both an input node (the name is prefixed with *i_*) and an output node (prefixed with *o_*. We need another event stating whether the prerequisites are fulfilled or not, modeled by the node *Prereq*. As we will show later, this will be used by the encapsulating action class to determine if the atomic action can be performed or not, so this must be an output of the atomic action class.

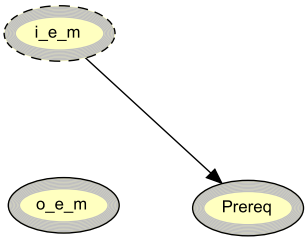

Figure 4: A class for the atomic action "M found on the victim" The special event output node *Prereq* has two states; *yes* and *no* for implying that the prerequisites for the atomic story are fulfilled or not. *i_e_m* is the input prerequisite event and *o_e_m* is the effect of the atomic story, and it is an output node.

Non-Atomic Actions

The representation of Non-atomic actions must ensure that any possible action it contains (called sub-actions) can be performed and also that sub-actions are performed until an end action has been performed. When and end sub-action has been performed, sub-actions can still be performed, as long as the last sub-action performed in the action is an end action. To handle this we make a representation of an action that will perform one of the possible sub-actions and make a several instances of it to perform several of the sub-actions. In the murder story all atomic actions are marked as endings of the action they are in, but several atomic actions can be performed in sequence, e.g. P4 "Ask A about C, where A first indicates a vague opportunity for C and then indicates a strong motive for C.

A Non-atomic action class has to make sure that one and only of its sub-actions is performed in each instance and that only possible sub-actions can be performed. This is done by using an instance of each sub-action class, i.e., the class representing the sub-action, an instance of the constraint class in Figure 5 for each sub-action. Evidence will be entered in the *Ok* nodes to make sure that only legal sub-actions can be performed. To ensure that at most one sub-action is performed, a variable with a state for each

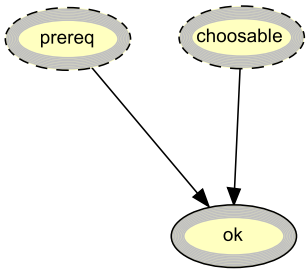

Figure 5: The constraint class that is instantiated once for each sub-action of an action

sub-action and one for nothing happening is created, and a variable for each sub-action monitoring if that sub-action can be performed created as a child of this. *Sc* is the variable describing possible sub-actions, each *Choosable* variable monitors if a sub-action is possible, they are *true*, if the corresponding sub-actions is possible. Note that *Sc* has Sc_ old as a parent, this is to ensure that if nothing happens in an action instance, nothing will also happen in subsequent action instances, of the current action. Also, *nothing happening* can only be occur if the last sub-action performed was an end sub-action or *nothing happening*. Making sure that an action ends with an end sub-action is done by introducing the variable *End* with two states, *y* and *n* as a child of *Sc* and letting the conditional probability table be constructed so *End* is in the state *y* if the sub-action performed in *Sc* is an end sub-action and *n* otherwise, and entering evidence on *y* in the last instance of the action class. Note that if *Sc* is in the state *nothing happening*, the last sub-action actually performed was an end sub-action, so *nothing happening* is also an end. In Figure 6 the OOBN unfolded to a Bayesian network for an action with two sub-actions, and one event, *e_m*, is shown, evidence will be entered in the two *Ok* variables to ensure that the *choosable* variables are only *true* if the prerequisites for the corresponding sub-action are met.

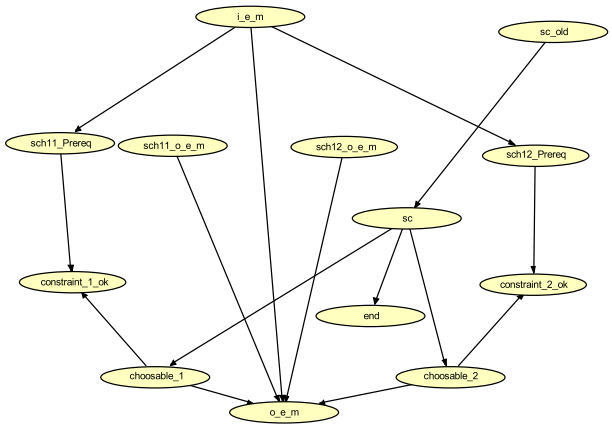

Figure 6: The part of an action class making sure that at most one sub-action is performed, and that only possible sub-actions can be performed. Note that the two atomic actions are called *Sch1.1* and *Sch1.2*.

Inputs and Outputs of an Action Class

All that is left for the action class is to make sure that the appropriate variables are available for the sub-actions, and that the past is updated according to the actions performed. This is done by having the union of inputs and outputs of the sub-actions as input to the action class, and the union of the outputs of these as output. The input of the action class ensures that any information that may be needed by a sub-action is available. The output of an action class ensures that any changes to events is available outside the action class instances. As only one sub-actions are performed in each instance, some of the events that can be changed may not necessarily be changed. In order for the action class to have the correct distributions in its outputs we need to know what these events are before the sub-action is performed. In Figure 7 a chapter class can be seen. Each output has the corresponding input as a parent to make sure that if the sub-action performed does not change the event, the events is not changed. Furthermore each output has the choosable variables corresponding to the sub-actions where they may be changed as parents, and the appropriate output of those sub-action instances are also parents. We have used parent divorcing to simplify the conditional probability tables, but as an example look at *o_e_m*, the output of the action class for the event *e_m*. It can be changed in sub-action 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 conditional probability table 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 on *choosable* variable can be true in each instance of the action class. Note that the action class should be instantiated several times where each output is is referenced by the corresponding inputs in the next instance.

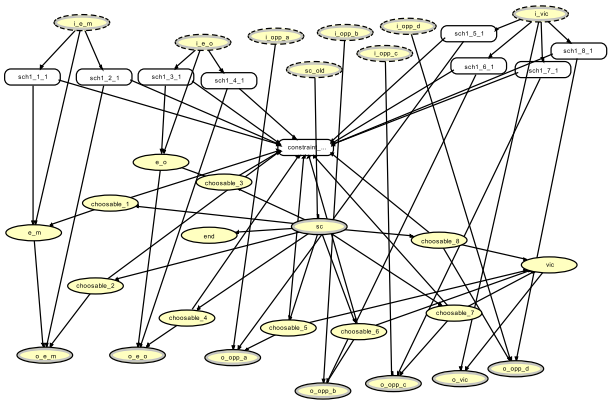

Figure 7: An action class for an action with eight sub-actions. The instances have not been expanded for overview.

Note that the sub-actions may themselves be non-atomic actions, and that the overall story will also be an action. An OOBN for the example can be generated automatically from the specification. We will call an OOBN generated in this way a NOOBN.

FUTURE RESEARCH DIRECTIONS

The work presented in this paper is still preliminary and subject to ongoing research. Given an NOOBN, we envision a game engine that, whenever it has to choose an action, ensures that the game generated will have an ending, regardless of the players actions. It furthermore has to ensure that the possible continuations of the game will be interesting, and that two games will not be identical even if the player always performs the same actions. There are three problems in constructing this game engine:

1. Ensuring diversity of games.
2. Guaranteeing that the last action of the game is marked as an ending of the game.
3. Generating interesting games.

Sampling the Actions

Whenever the game engine chooses an action it will start by entering evidence on the actions already performed in the *Sc* nodes of the NOOBN. We will then sample possible continuations of the game to make it likely that we get different games, 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 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. We will then sample the next *Sc* variable given the previous actions. This however requires a propagation of the network as we have evidence both before this point of the game (the previous actions) and later (we want all actions to end with an action that is marked as an ending. We will continue to sample *Sc* variables as long as we have not reached the end of the overall action. We have reached the end if there are no more unsampled *Sc* nodes. All the *Sc* nodes will now be a complete game, with an ending. Note that we are also sampling the player actions.

Rating the Game

When choosing the next action, we want to ensure that it leads to interesting games. This requires a method of scoring a game.

A simplistic approach is to associate a score value with each atomic action. The measure the value of a story we can either accumulate the values or compare the values to some distribution (such as the tension value arc in [Mateas and Stern, 2003]. Such a simplistic approach will not capture all the intricacies of a good game, e.g. in a murder story it is 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. Developing useful algorithms to score games based on the narrative content and other factors is a subject of future research.

Assuming games can be scored, the NOLIST game engine facilitates the rating of each possible next action by adding the weighting the score of each sampled game resulting from that action with its probability. The next action can then be chosen among the possible next actions based on their rating. The chosen action is then performed and the past updated by it effects, and the process is repeated until the story ends.

REFERENCES

[Bangsø and Wuillemin, 2000] Bangsø, O. and Wuillemin, P.-H. (2000). Top-down construction and repetitive structures representation in Bayesian networks. In *Proceedings of the Thirteenth International FLAIRS Conference*, Florida, USA.

[Jensen, 2001] Jensen, F. V. (2001). *Bayesian Networks and Decision Graphs*. Springer Verlag, New York.

[Juul, 2004] Juul, J. (2004). *Half-real. Video games between real rules and fictional worlds*. PhD thesis, IT-Universitetet.

[Koller and Pfeffer, 1997] Koller, D. and Pfeffer, A. (1997). Object-oriented Bayesian networks. In Geiger, D. and Shenoy, P. P., editors, *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence*, pages 302–313, San Francisco. Morgan Kaufmann Publishers, San Francisco.

[Mallon and Webb, 2000] Mallon, B. and Webb, B. (2000). Structure, causality, visibility, and interaction: propositions for evaluating engagement in narrative multimedia. *International Journal of Human-Computer Studies*, 53:269–287.

[Mateas and Stern, 2003] Mateas, M. and Stern, A. (2003). Facade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference*.

[McKee, 1997] McKee, R. (1997). *Story: Substance, Structure, Style and The Principles of Screenwriting*. HarperCollins, New York.

[Pearl, 1988] Pearl, J. (1988). *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers.

[Szilas, 2003] Szilas, N. (2003). IDtension: a narrative engine for interactive drama. In *1st International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE 2003), March 24-26, 2003, Darmstadt, Germany*.

APPENDIX

In the following table each number defines an atomic action. The last line of each entry contains the effects while the remaining lines are the prerequisites.

1.1.1.1	No evidence for an opportunity is known M indicating opportunity is found
1.1.1.2	No evidence for an opportunity is known N indicating opportunity is found
1.1.2.1	No evidence for a motive is known X indicating motive is found
1.1.2.2	No evidence for a motive is known Y indicating motive is found
1.1.3.1	It is not known who found the victim A found the victim, A had vague opportunity
1.1.3.2	It is not known who found the victim B found the victim, B had vague opportunity
1.1.3.3	It is not known who found the victim B found the victim, C had vague opportunity
1.2.1.1	It is not known if B had a motive B has a motive
1.2.1.2	It is not known if B had an opportunity B had vague opportunity
1.2.1.3	B had vague or clear opportunity A had no opportunity, B had no opportunity
1.2.2.1	X indicating motive is found C had strong motive
1.2.2.2	It is not known if C had an opportunity C had vague opportunity
1.3.1.1	It is not known if A had a motive A had weak motive
1.3.1.2	A had vague opportunity, N indicating opportunity is found A had clear opportunity
1.3.2.1	B had vague opportunity, M indicating opportunity is found, it is not known if B had a motive B had clear opportunity
1.3.3.1	It is not known if C had a motive C had weak motive
1.3.3.2	C had vague opportunity, N indicating opportunity is found C had clear opportunity
1.3.3.3	C had vague or clear opportunity B had no opportunity, C had no opportunity
1.4.1.1	X indicating motive is not found Y indicating motive is found
1.4.1.2	A had weak motive, X indicating motive is found A had strong motive
1.4.1.3	It is not known if A had an opportunity A had vague opportunity
1.4.2.1	It is not known if B had a motive B had weak motive
1.4.2.2	B had vague opportunity B had clear motive
1.5.1.1	A had strong motive, A had clear opportunity
1.5.1.2	B had strong motive, B had clear opportunity
1.5.1.3	C had strong motive, C had clear opportunity