# Automatically Generating Summary Visualizations from Game Logs

**Yun-Gyung Cheong[*], Arnav Jhala[†], Byung-Chull Bae[†], and R. Michael Young[†]**

[†]Liquid Narrative Group, Department of Computer Science, North Carolina State University
[*]AMC Part, Graphics&OS Group, SAIT, Samsung Electronics, Co., LTD.

**Abstract**

In this paper we describe a system called ViGLS (**Vi**sualization of **G**ame **L**og **S**ummaries) that generates summaries of gameplay sessions from game logs. ViGLS automatically produces visualization of the summarized actions that are extracted based on cognitive models of summarization. ViGLS is implemented using a service-oriented architecture, de-coupling the summarization methods from any particular game engine being used. The camera code libraries used in visualization are based on constraint based camera control approaches and, in our implementation, make use of the scripting layer of the Unreal game engine.

## 1. Introduction

Advances in game hardware and market demand are motivating game makers to expand the narrative content and style of interaction within new game titles. For most players, the process of playing a game is broken across many relatively short sessions. Players of single player games often play a single game over the course of weeks or months, saving game state at the end of each session and continuing from the saved state upon returning to the game later. Players of Massively Multi-player Online Role Playing Games (MMORPGs) also participate in gameplay over extended periods of time (though in their cases, game play may continue in their absence). It has been reported that players can become extremely involved in these virtual worlds for extended periods of time (Griffiths et al., 2003).

As gameplay sessions become shorter relative to the overall length of a game and the complexity of the interaction within game environments increases, a summary of past gameplay can make the gameplay more enjoyable to the player in single player games. Further, game summaries provided to players in multi-player persistent world games could help players maintain context and engagement in their games during times when they are not logged in.

In the work we describe here, we use a 3D game engine to visualize a summary of game play derived from game logs. This visual approach to summarization may be more accessible for gamers used to experiencing the rich 3D interfaces when interacting with their game worlds. Further, since our system generates its summaries using the same engine where the original game engine occurred actions can be recreated (almost) exactly as they were executed.



## 2. Related Work

*Story Summarization:* The problem of creating concise textual summaries of stories has been addressed by several researchers (Capus and Touringy 2003; Lehnert 1981). The GARUCAS framework, developed by Capus and Touringy (2003), summarizes a new story using stored cases representing stories and their summarization methods. When no directly matched cases are found for an input story to be summarized, GARUCAS builds a structure composed of plot units that describe the change of emotional states for the characters. Then, a case that shares the largest parts in structure with the input story is selected and the system utilizes its summarization method to extract core events from the input story. As a result, the system produces a qualitatively effective summary when a similar case to the input story is retrieved. When such cases are not found, however, GARUCAS requires manual annotation of the input story for the system to identify plot units.

In contrast, automated commentary generation systems (Tanaka-Ishii et al. 1998; Voelz et al. 1999; André et al. 2000) take as input raw log data and produce textual descriptions of action. Their objective, however, is to produce individual text components that describe each action in the log; they do not focus on managing the coherence of the overall summary text output.

*Video Summarization:* As an effort to provide online summarization of a conventional television drama, Jung et al. (2007) describe the Narrative Abstraction Model, used to extract significant scenes from a video source by analyzing the changes in each shot's visual style (e.g., lighting, color contrast, shot edits, spatial and temporal relationships). Although the model provides a complete solution to extracting a narrative-based summary from a raw video clip, its focus on visual editorial techniques as keys to signal scene significance may limit its applicability in media where these visual editorial are absent. To overcome these problems, we present a generic narrative summarization model that utilizes task-based knowledge (i.e., causal relations) of a given gaming experience.

*Game Log Summarization:* To enhance the functionality of narrative summarization of game experiences we draw

upon research in cognitive and computational models of narrative summarization. This work builds on previous text-based approaches to game log summarization (Cheong and Young 2006) by extending the model for selecting important events from the game logs. Further, we utilize the rendering capabilities of game engines to re-create sequences of actions, providing effective summary videos to players.

In this work, we assume that actions included in game logs collectively represent goal directed behavior on the part of the player. We restrict our discussion here to game logs that are created by players following the rules of the game and actively trying to achieve the game objectives. Players may execute actions that do not eventually contribute to the objective of the game during their exploration process. All these actions are included in the game logs. From these actions, a set of salient events can be identified by their causal and temporal relationship to the goal of the plans (that correspond to game objectives).

We rely upon cognitive models of story recall (Trabasso and Sperry 1985) and question-answering in the context of stories (Graesser et al. 1991) to determine the importance of salient actions found in game logs. Intelligent camera control systems (Jhala and Young 2006) on game engines provide the framework for controlling game cameras to satisfy given viewing constraints. This technology is used here to manage execution and visualization of summarized actions. Our approach involves three tasks: translating a game log into a plan structure, constructing a summary centered on important events in a coherent manner, and executing the actions in the resulting summary with automated camera control for visualization. These tasks are incorporated in a framework called *ViGLS* (**Vi**sualization of **G**ame **L**og **S**ummaries – pronounced: *wiggles*).

## 3. ViGLS: Visualization of Game Log Summaries

In this section we present *ViGLS*, a framework for summarizing game experiences as narratives. As illustrated in Figure 1, *ViGLS* has a pipeline architecture consisting of three components: a log analyzer, a summarizer, and a visualizer. The log analyzer takes a game log as input and generates a sequence of actions structured as a totally ordered plan that achieves the immediate objectives of the game. The summarizer utilizes the resulting plan data structure to identify essential elements that can be included in an effective summary of the game's events. Finally the summarized narrative is sent to the visualizer that adds constraints for the in-game virtual camera for visualizing the summary actions. An execution manager in the game engine manages the execution of summary actions as well as camera control. The following subsections discuss each component in detail.

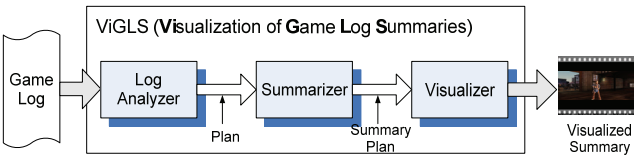


Figure 1. Pipeline Architecture of ViGLS

### 3.1 The Log Analyzer

Text files that record events in a game engine during player's gameplay sessions are called game *logs*. Traditionally, logs contain administrative information and game state messages, although there is some consideration of expanding the typical information they track (Garner 2004). In this work we create custom logs that record time stamped actions that players execute including relevant game state information. This representation provides a textual (ASCII) characterization of a player's activity during an entire gameplay session.

*Plan* data structures in artificial intelligence typically represent sequences of actions and the causal and temporal relationships between them (Weld 1994). Each action in a plan is called a *step*. The enabling conditions of a step are called *preconditions*. The conditions in the world changed by a step's execution are called the step's *effects*. Any two steps in a plan are connected by a *causal link* just when an effect of the first step enables a precondition of the later step. In order to better reason about relationships between player's actions and to determine their relative importance, the actions present in the logs are converted to a plan data structure. Each action that is present in the log is added to a partial plan by instantiating a plan step. For instance, consider the log entry `T4: Barman gave Lane Booze`. This is matched with a plan step template in the action library of the system. This template is given here:

**Action:** Give
**Parameters:** ?giver, ?taker, ?thing
**Constraints:** (person ?giver)(person ?taker)(thing ?thing)
**Preconditions:** (has ?giver ?thing)
**Effects:** (has ?taker ?thing) (not (has ?giver ?thing))

For this research, we use the plan structure generated by the Longbow planner, a hierarchical, partial-order causal link planner (Young et al., 1994)[1]. In our current work, a plan is represented as a totally ordered series of plan steps; assuming that action descriptions written in source log files appear in the actual order of execution, plans resulting from the log files' transcription should be directly executable within the game environment.

**Converting Game Logs into Plan Data Structures**

We illustrate the plan building process through an example shown in Figure 2, drawn from a domain called WestWorld executing in the Unreal game engine. In this

[1] For brevity, we discuss only a non-hierarchical version of Longbow here.

particular example, the user plays the character Lazarus Lane and is given the objective of getting a bottle of booze from the local bartender. The player first moves to the bank during the exploration process, not knowing the location of the bar. Then the user threatens the bartender with a gun (as her character does not have money to buy the booze) and eventually obtains the bottle. Given the log of these events, the log analyzer first converts the log messages into a sequence of instantiated actions based on a library of action class templates similar to the one shown above.

Next, the system creates an empty plan *P* with its first step *INIT*, describing the initial game world as its effects, and the last step *FINISH*, describing the goal state as its preconditions. It then inserts the instantiated actions from the log into his plan structure, ordering them between the *INIT* and *FINISH* steps. Finally, the system establishes causal links to establish the preconditions of every step in *P*. A causal link is created for each unifying effect-precondition pair of actions in *P*. This is done through backward chaining from the *FINISH* step to the *INIT* step. After adding causal links for all of the plan's steps' preconditions, the system reviews each step and deletes any that do not contribute at least one causal link drawn from an effect. The complete plan *P* (as the diagram in Figure 3) is passed to the summarizer that is described in the next section.

T1: Player "Lane" moved from "Bank" to "Bar".
T2: Player "Lane" looked at "Barman".
T3: Player "Lane" threatened "Barman" with "Gun".
T4: Character "Barman" gave "Lane" "Booze"

Figure 2. A game log example

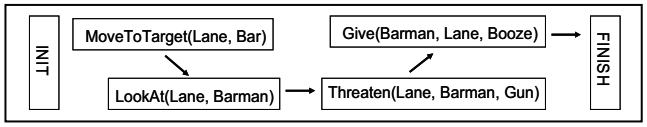


Figure 3. A plan that is converted from the game log

## 3.2 The Summarizer

The summarizer performs two tasks on the plan generated by the log analyzer: *kernel extraction* and *coherency checking*. Kernels refer to essential story events that cannot be eliminated from a potential summary without harming the reader's story understanding (Chatman 1978). As illustrated in Figure 4, the summarizer is composed of three subcomponents: a) the kernel extractor extracts kernels from the given plan by measuring the importance or weight of each plan step, b) the coherency evaluator checks the coherency of a potential summary, and c) the viewer model characterizes a viewer's preference and elements of his or her narrative comprehension process.

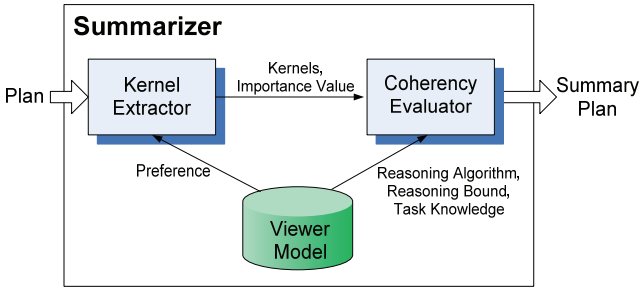


Figure 4. Architecture of the Summarizer

### The Kernel Extractor

The kernel extractor determines which story events (or steps) in a plan are most important for the story's summary and uses this evaluation to construct kernels. To select important story events for kernels, the kernel extractor rates the significance of each step in the story plan by taking into account two factors: (1) the causal relationships of each action with other actions and the goal state and (2) the viewer's preference in the viewer model. The relative weights of these two factors are controlled by coefficients that can be set by the system designer. In this section, we describe the role of each coefficient and its impact on the summary. Specifically, two coefficients associated with characters and items are particularly relevant to game environments.

Our approach to analyze the causal relationship among story events is based on the causal chain network model for the story recall, devised by Trabasso and Sperry (1985). This model asserts that the number of direct causal connections into and out of a story action is closely related to a reader's recall of that action and his or her subjective judgment of its importance to the story. Using the causal network model as a starting point, we define three types of elements to characterize events in a story plan: the opening act, the closing act, and the motivating act. Opening acts are the first actions in the story – those that connect propositions from the initial state to later events; closing acts are the last actions that occur in the story; motivated acts are plan steps that directly connect to the goal state.

As the summarizer takes a plan-structured log from the log analyzer, it analyzes the characteristics of the plan structure in terms of the causal relationships between steps, each character's importance, and each item's importance. The current version of the summarizer calculates the importance of characters and items based on the frequency with which the character or items play a role in the story's actions relative to the overall set of events in the story.

After the kernel extractor finishes the analysis of causal relationships, character importance, and item importance, it calculates *w(a)*, the final importance of step *a*, using the following equation:

$$w(a) = k_c^{CC(a)} \cdot \left(k_i IN(a) + k_{in} INIT(a) + k_o OUT(a)\right) + k_{ch} CH(a) + k_{it} IT(a) \quad (1)$$

Here *IN(a)* returns the number of incoming causal links to step *a* except the links that originate from the initial state; *INIT(a)* returns the number of incoming causal links from the initial state to step *a*; *OUT(a)* returns the number of *a*'s outgoing causal links when *cc(a)* returns the causal chain value of *a*; $k_i$, $k_{in}$, $k_o$, and $k_c$ are coefficients for causal relationships; $k_{ch}$ and $k_{it}$ are coefficients for character importance and item importance; *CH(a)* and *IT(a)* return the character importance and the item importance respectively; *CC(a)* represents the causal chain value of an event that is determined by the event's causal chain type.

In this formula significant categories, such as motivated acts, are assigned higher integer values in order to increase the likelihood that they will be selected as kernels. The particular values for these coefficients can be determined empirically. For instance, to increase the contribution of causal relationships to the summary, the coefficients $k_i$, $k_{in}$, $k_o$, and $k_c$ can be set to any positive real numbers greater than 1. In contrast, setting these coefficients to real numbers between 0 and 1 will reduce their effects on the summary.

After computing the importance of each story event, the top *N* events are identified as kernels. The value for *N* can be set by the user as a desired story length, or it can be calculated from a predefined ratio against the total number of actions in the plan. The kernel extractor sends these *N* kernels and their importance values to the coherency evaluator.

**The Coherency Evaluator and the Viewer Model**

Studies suggest that plan-related reasoning process in humans can be modeled effectively by partial-order planning algorithms (Rattermann 2001) and that refinement search (Kambhampati *et al.* 1995) – the plan construction process performed by the planning system used here – can be used essentially as a surrogate to characterize a user's plan reasoning process (Young 1999). To determine if a given set of kernels will appear to a user as a coherent story summary, we exploit a plan-based model of the user's anticipated comprehension process when viewing the candidate summary. This viewer model is composed of a reasoning algorithm, a reasoning resource bound, task knowledge describing the story world's domain, and a representation of the user's preferences for action sequences. The model is implemented using the Longbow planning system (Young *et al*. 1994).

To evaluate a set of candidate kernels, the viewer model uses its planning algorithm to re-create a complete set of plan actions that lead to achievement of the story objectives (including less important but causally relevant actions). If a complete plan can be constructed from the kernels, we rate the set of kernels as coherent. Otherwise, the coherency evaluator adds actions to the candidate set in the order of importance until it finds a set of kernels that lead to a complete plan, at which point it terminates.

## 3.3. The Visualizer and the Execution Environment

The summary of actions selected by the summarizer is visualized by:

1) Selecting appropriate camera placement parameters for viewing the actions in the summary plan (essentially choosing camera shots).

2) Taking character actions from the summary plan and executing them in a game world, concurrently maintaining the camera positions to film the actions and avoid occlusions with scene geometry.

Task 1 is carried out by a shot planner and Task 2 is handled by an execution manager implemented on the Unreal game engine.

*Shot Planner:* The action summary generated by the skeleton builder is a plan data structure specifying, among other details, the sequence of summarized actions. This action sequence is input to a camera shot planner (Jhala and Young 2006) that determines appropriate shot types and shot sequences needed to film the input actions. The camera planner's directives maintain focus of the camera on the salient elements of each action in the summary and specify appropriate camera visualization parameters. The parameters for camera actions are bound through constraints that refer to the story actions filmed by the camera. Each camera action starts at the beginning of the execution of the relevant story action and maintains focus till the beginning of the next action. The plan operator for a camera's *track* action is shown below.

**Operator:** Track-Actor-Absolute
**Description:** Tracks an object at an absolute distance and angle.
**Parameters:** ?obj, ?shot-type, ?shot-angle,
?tstart, ?tend, ?tsetup, ?teasein
**Preconditions:** (focus ?obj ?shot-type ?shotangle)@[?tstart)
**Constraints:** (> 10 (- ?tend ?tstart)) (agent ?step ?obj)
**Effects:** (tracked ?obj ?shot-type ?shotangle)@[?tstart, ?tend)
(Bel V (Occurs ?act))

Each story action in a summary is also associated with a text action that executes as a subtitle describing the action. These subtitles are built using simple text templates that are filled in by the parameters bound by the story action. The summary visualization is executed and recorded using the Zocalo service-oriented architecture for interactive storytelling (Young *et al.* 2004). An application written in C# schedules the execution of actions by dispatching them in the order specified by the planning algorithm, sending SOAP commands to control the game engine via a socket connection.

*Execution Manager:* An action executor process implemented on the game engine reads the SOAP plan and runs methods corresponding to each of the plan's actions within the game engine (each operator in the story plan is represented by an action method with matching

parameters). Each action method contains functions for checking preconditions, executing the body of the action, and verifying that effects of the action hold after it completes. The general action class signature is as follows:

```
Class Track-Actor-Absolute extends ZAction_ZCamera
   Actor ObjectOfAttention;
   …. Other Parameters …
   function checkPreconds() { … }
   state executing {
         function tick {
            maintainCameraConstraints();
          }
   }
   function assertEffects() { … }
   function markasexecuted{ return bSuccess; }
```

Action classes on the game engine are of three types: *controller action classes* control character actions in the game environment and are responsible for playing appropriate character or object animations, *sub-title action classes* execute as informational text overlays on the screen and use a fixed set of text-templates, *camera action classes* execute by updating a global set of geometric constraints on the properties of the game engine's camera.

An automated cinematographer continually computes camera positions to satisfy the current set of constraints set by the executing camera action classes. This process executes in real-time and ensures occlusion free viewing of actions. Our execution management module is based on the Mimesis architecture developed by (Young *et al.* 2004). The algorithm for planning camera movements and their execution is described in more detail by Jhala and Young (2006).

## 4. Example

This section presents an example of a visualized summary of a game experience produced by ViGLS. Section 4.1 describes a text summary generated by the summarizer, taking as input a plan-structured game log that is the output of the log analyzer. Section 4.2 shows a visualization of the summary generated in Section 4.1.

> [15] Lane moved to the bank. [14] Lane moved to the bar. [13] Lane looked at the barman. [12] Lane threatened the barman with a gun. [11] The threatened barman gave Lane a bottle of booze that he had. [10] Lane moved to the bank. [9] Lane moved to the smithy. [8] Lane looked at the blacksmith. [7] Lane bribed the blacksmith with the booze with him. [6] The bribed blacksmith gave Lane a machete that he had. [5] Lane moved to the bank. [4] Lane looked at the teller. [3] Lane bribed the teller with the machete. [2] The bribed teller gave Lane gold that she had.

Figure 5. Story created by Longbow realized into text

### 4.1. Summarization

The output of the summarizer is a set of important actions selected from the plan-structured input log data. As shown in Figure 5, we realized the input story plan into text using simple templates (here one sentence represents a single action in the plan). In the story the protagonist Lane plans to threaten a barman, bribe a blacksmith, and bribe a teller to get gold in the bank.

As described above, manipulating coefficient values in the summarizer setting will bring in different summary results. For instance, with default coefficient settings – where $k_i$= 3.0, $k_{in}$=1.0, $k_o$=3.0, $k_c$=1.0, $k_{ch}$=0.5, and $k_{it}$=0.3; Opening/Closing factors=1.0; Motivated factor=2.0 –, the summarizer determines actions #2, #3, #7, #6, and #9 (#2 is the action with the highest importance value) as kernels consisting of five actions. With the same factors except the modification of $k_{in}$ (the coefficient associated with the number of incoming causal links from the initial state) from 1.0 to 3.0, the summarizer determines actions #2, #3, #6, #7, and #12 as the kernels. Future studies will seek to determine how to set these values to reflect viewers' preferences.

### 4.2. Visualization of the Summary

As explained in Section 3.3, a visualization of the summary is created by sending the summary plan – including character actions, text actions, and camera actions — to the game engine's execution environment. Figure 6 presents screenshots taken from the output of the visualization. Due to space limitations, subtitles in these screenshots are too small to be read here. In summary, Plate 1 in the picture shows Lane threatening the bartender. Plate 2 on the right shows the bartender giving the bottle to Lane. In Plate 3, Lane is seen leaving the bar with the bottle. The rest of the plates show Lane at the bank bribing the teller to get gold.

## 5. Discussion

This paper describes a framework for summarizing game experiences by translating a game log into a plan data structure, then extracting essential events from the plan based on their causal relationships to the story goals. Summaries created by the algorithm are based on cognitive models of narrative comprehension. The algorithm can be applied to different domains by manipulating the co-efficient values appropriate for each domain.

We build on our previous work developing methods for executing game engine events and controlling a 3D camera dynamically to create effective videos for the generated summaries. In future work, we plan to conduct user studies to evaluate the effectiveness of summaries presented in a visual form compared to traditional text based summaries. Our assumption is that game players will prefer visual summaries with descriptive sub-title text over text-only summaries.

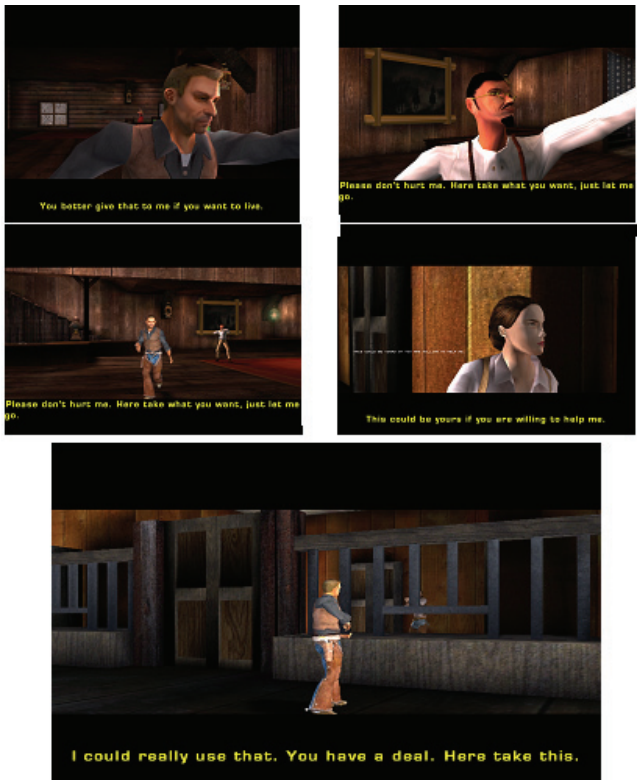


Figure 6. Automatically generated summary visualization

This work advances previous work on game log summarization with two main contributions. First, ViGLS generates summaries in the visual medium by procedurally re-creating the summary on the game engine. Second, we describe in detail the impact of various coefficients in the summarization algorithm and extend the summarization model to include additional coefficients that represent the importance of characters and objects in the game.

## 6. Acknowledgements

The authors wish to thank the Lawdogs Unreal mod team for use of the Lawdogs mod in the construction of the WestWorld game environment. Portions of this work were supported by NSF CAREER Award # 0092586 and Award #0414722.

## References

André, E., Binsted, K., Tanaka-Ishii, K., Luke, S., Herzog, G., and Rist, T. 2000. Three RoboCup Simulation League Commentator Systems. *AI Magazine* 21(1):57-66.

Capus, L. and Tourigny, N. 2003. A case-based reasoning approach to support story summarization. *International Journal of Intelligent Frameworks* 18: 877-891.

Chatman, S. 1978. *Story and Discourse: Narrative Structure in Fiction and Film*. Ithaca, NY: Cornell University Press.

Cheong, Y. and Young, R. M. 2006. A framework for summarizing game experiences as narratives. In Proceedings of AIIDE-06.

Graesser, A.C., Lang, K.L., & Roberts, R.M. 1991. Question answering in the context of stories. *Journal of Experimental Psychology: General* 120(3): 254-277.

Garner, S. 2004 June 24. Half-Life Standard Log Format Specification version 1.03. < http://www.hlstats.org/logs/>. Accessed 2006 March 24.

Griffiths, M. D., Davies, M.N.O. & Chappell, D. 2004. Online computer gaming: a comparison of adolescent and adult gamers. *Journal of Adolescence* 27: 87-96.

Jhala, A. & Young, R. M., Representational requirements for a plan based approach to automated camera control. 2006. In Proceedings of AIIDE-06.

Jung, B., et al. 2007. A narrative-based abstraction framework for story-oriented video. *ACM Transactions on Multimedia Computing, Communications and Applications* 3(2), Article No.11.

Kambhampati, S., Knoblock, C. A., and Yang, Q. 1995. Planning as Refinement Search: A Unified Framework for Evaluating Design Tradeoffs in Partial-Order Planning. *Artificial Intelligence* 76(1-2): 167-238.

Lehnert, W.G. 1981. Plot units and narrative summarization. *Cognitive Science* 5(4): 293-331

Rattermann, M. J., Spector, L., Grafman, J., Levin, H. and Harward, H. 2002. Partial and total-order planning: evidence from normal and prefrontally damaged populations. *Cognitive Science* 25(6): 941-975.

Tanaka-Ishii, K., Noda, I., Frank, I., Nakashima, H., Hasida, K., and Matsubara, H. 1998. 'MIKE: An automatic commentary framework for soccer. In Proceedings of ICMAS, 285– 292.

Trabasso, T. and Sperry, L.L. 1985. Causal Relatedness and Importance of Story Events. *Journal of Memory and Language* 24: 595-611.

Voelz, D., André, E., Herzog, G., and Rist, T. 1999. Rocco: A RoboCup Soccer Commentator Framework. M. Asada and H. Kitano (eds.): RoboCup-98, *LNAI* 1604: 50-60, Springer-Verlag Heidelberg Berlin.

Weld, D. An Introduction to Least Commitment Planning. 1994. *AI Magazine* 15(4): 27-61.

Young, R.M., Pollack, M.E., and Moore, J.D. 1994. Decomposition and causality in partial-order planning. In Proceedings of AIPS-94, 188-194.

Young, R.M. 1999. Using Grice's Maxim of Quantity to Select the Content of Plan Descriptions. *Artificial Intelligence* 115(2): 215-256.

Young, R.M., Riedl, M., Branly, M., Jhala, A., Martin, R. J. and Saretto, C.J. 2004. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development* 1(1).