

Talking with NPCs: Towards Dynamic Generation of Discourse Structures

Christina R. Strong and Michael Mateas

Expressive Intelligence Studio (EIS)

University of California, Santa Cruz

Santa Cruz, California, USA

{crstrong,michaelm}@cs.ucsc.edu

Abstract

Dialogue in commercial games is largely created by teams of writers and designers who hand-author every line of dialogue and hand-specify the dialogue structure using finite state machines or branching trees. For dialogue heavy games, such as role playing games with significant NPC interactions, or emerging genres such as interactive drama, such hand specification significantly limits the player's interaction possibilities. Decades of research on the standard pipeline architecture in natural language generation has focused on how to generate text given a specification of the communicative goals; one can imagine beginning to adapt such methods for generating the lines of dialogue for characters. But little work has been done on the problem of procedurally generating dialogue structures, that is, dynamically generating dialogue FSMs or trees (more generally, discourse managers) that accomplish communicative goals. In this paper we describe a system that uses a formalization of backstory, character information, and social interactions to dynamically generate interactive dialogue structures that accomplish desired dialogue goals.

Introduction

Non-player character (NPC) dialogue is a significant component of many contemporary games; story progression, character relationships, and backstory and world information is often presented through interactive dialogues. Even games such as *Mass Effect*, in which the designers tried to *reduce* interactive dialogue and depend more on cinematics to convey story and character progressions, still had approximately 28,000 lines of conversation dialogue, averaging 12.5 words per line (Walters and Pressey 2008).

Game dialogue is currently created by teams of writers and designers who hand-author every line and hand-specify the dialogue structure using finite state machines or branching trees. In this paper, we report on the design and initial results of a system that procedurally generates dialogue finite state machines (FSMs) based on a declarative model of the information possessed by NPCs and their emotional inclinations. The system could be used offline to automate

the creation of dialogue FSMs, or potentially, online, to create dialogue structures tailored to evolving game state of a particular play session. The emerging genre of interactive drama has utilized the most advanced dialog structures, as the primary gameplay is dialogue-based. Thus, we are using the dialogues in the interactive drama *Faade* (Mateas and Stern 2003) to inform our model of dialogue structure generation.

Player interaction in *Faade* is organized around social games, "head games" the player plays with the NPCs through discourse actions (Mateas and Stern 2005b). For example, one of the social games in *Faade* is the affinity game, in which the NPCs bring up an emotionally charged issue, argue about it, and force the player to take sides in their argument. The affinity each NPC feels towards the player is influenced by whom the player takes sides with. In the course of the argument, the player also learns backstory information about the characters and their relationships. In our work we start with the affinity game as the first social game to procedurally model.

The social games in *Faade* are organized around dramatic beats (McKee 1997), the smallest units of dramatic action that change character relationship and story state. In *Faade* the behaviors for beats are hand-authored. Though the reactive execution of the beat will result in many variants as the player interacts in the beat, the decisions about which information will be revealed in the beat, how the characters will counter each other in the argument, and how the characters will force the player to take sides, are decided at design time by the authors. There is nothing in the system that declaratively understands what an "affinity game" is, nor how the information and emotions revealed in the beat relate to the social game. The goal of our work is to explicitly represent backstory, character and relationship information, as well as to procedurally model the dynamics of social games, so as to dynamically generate the dialogue structure of the social games instantiated in beats. This provides significant authoring leverage; given a fixed quantity of possible dialogue content, it can be sliced and diced *many* possible ways into dialogue structures. This allows the player to traverse many more individual trajectories through the content than is possible if the content is manually organized by authors into a relatively small number of dialogue structures.

In this paper we describe a system for procedurally gen-

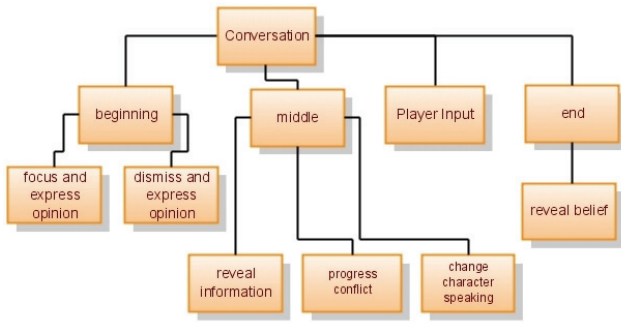


Figure 1: Model of a conversation

erating dialogue structures; that is, given the system’s high-level communicative goals for an NPC interaction (ie. which backstory information or topics should be revealed and how the social and emotional states of NPCs should change during the conversation), dynamically generating a dialogue FSM (more generally, a discourse manager) that accomplishes these goals. In the rest of this paper we describe related work, present our architecture, describe an example dialogue generated by our architecture, and discuss lessons learned and issues with this approach.

Architecture

Figure 1 depicts the hierarchical structure of an affinity game. An affinity game consists of a character bringing up a controversial topic, several argumentative exchanges about this topic, the player being asked to take sides, and a final revelation as a function of whom the player took sides with. In the middle of the conversation, NPCs reveal information and progress the conflict, while taking turns between the characters involved in the argument. In the final revelation, an NPC reveals some portion of his or her beliefs about the underlying conversational theme. Note that the player might interact with the NPCs during the argument, and not just at the point where they take sides. Such interaction might change the focus of the argument, or mix in a peripheral topic. However, for the purposes of our initial system, we are focusing on the simplified case where the primary player interaction occurs after the progression of the conflict.

Since a conversation lends itself well to hierarchical decomposition, we chose a hierarchical task network (HTN) planner to create dialogue structures. HTN planning takes an initial world-state and creates a plan that accomplishes specified tasks. This is done by decomposing tasks into smaller tasks, until the subtasks can be accomplished by planning operators. Authors provide the initial world state, which in our case includes assertions about conversational topics and characters. The tasks are social game author goals, such as revealing information about a specific story theme. Given a task and operator description of a social game, many dialogue structures can be created by changing the assertions about topics and characters. We use the SHOP2 planner, (Karlsson 2001), a partial order HTN planner developed by the University of Maryland.

In addition to the natural fit between the hierarchical structure of social games and HTNs, we selected the HTN planning formalism because of its forward state progression. This allows us to easily implement functional effects, such as incrementing and decrementing tension, social relationship state, etc. We found such functional effects necessary for dialogue structure planning.

Next, we describe our domain model for representing conversational content for affinity games, our task network for modeling the logic of affinity games, and our approach for interpreting a linear HTN plan as a branching dialogue FSM.

Domain Description

Conversational content is organized into themes, foci, past events, and information related to past events. Themes are underlying sources of conflict for which characters may have strong, opposing beliefs. A focus provides an entry point into a theme, a concrete object or issue which can segue into a theme, and provide a focus for a subsequent argument. Characters have emotions about the focus; a focus for which characters have conflicting emotions is a good candidate for an affinity game argument. Past events provide backstory that characters can use to support their positions. Associated with past events are specific pieces of information about the event, as well as emotions associated with the different pieces.

The setting for our prototype content pool is a playground, where two children, Meg and Chuck, are having an argument. There are two underlying story themes. The first theme is a personality conflict where Meg thinks Chuck is too relaxed and laidback while Chuck thinks Meg is too uptight and needs to enjoy life more. The second theme revolves around class, where Chuck thinks Meg’s family is richer, while Meg thinks the two families are the same. These themes are introduced via one of two foci, either a new video game that is coming out or a violin that Meg is carrying with her. One focus is brought up by one of the characters in order to start a conversation. The conversation then progresses to arguing about the topic at hand, with Meg and Chuck supporting their respective positions using information from events that have happened in their mutual past. The argument grows until one of them forces the player to choose sides, resulting in a revelation from one of the characters about his/her belief about the current story theme.

Information in the content pool is encoded as logical predicates. For example, to describe a character’s emotion associated with a thematic focus, we use an *emotion* predicate that takes a character, the name of a focus, and an emotion (some example emotions are excited, happy, frustrated, and annoyed) as arguments. So, to represent that Chuck is excited about a new videogame, which is one focus of conversation, we write: (emotion Chuck new-video-game excited).

Information about story themes, beliefs, people, and past events are similarly encoded, forming a network of connections within the content pool. Everything relates back to a story theme, since these are what drive the conversation. Figure 2 shows the representations of the beliefs, past events, and foci related to the personality conflict theme. The information associated with past events can be brought up by

```

(person Meg) (person Chuck) (focus new-video-game)

(story-theme playfulness-personality-conflict)
(belief Meg live-by-the-book)
  (related-belief
   playfulness-personality-conflict Meg live-by-the-book)
(belief Chuck have-fun-in-life)
  (related-belief
   playfulness-personality-conflict Chuck have-fun-in-life)

;; Chuck's point of view
(pastevent allowanceChuck)
(primary-person allowanceChuck Chuck)
;; Meg was directly involved
(others-involved allowanceChuck Meg)
;; directly related to the Meg allowance past event
(related-pastevent allowanceChuck allowanceMeg)
;; pieces of information about the event
(information allowanceChuck borrowed-from-Meg)
;; both grateful and embarrassed he had to borrow
(emotion Chuck borrowed-from-Meg embarrassed)
(emotion Chuck borrowed-from-Meg gratitude)
(information allowanceChuck had-spent-his-on-video-games)
;; is content with spending his money on what he wanted
(emotion Chuck had-spent-his-on-video-games satisfaction)

```

Figure 2: Personality Conflict Predicates

characters in the course of the arguments. This structure allows the planner to reason over important information in the story, relating everything to the current story theme.

Note that in the domain description we use long symbols such as *had-spent-his-on-videogames*. The reader may wonder how non-composition symbols such as these acquire meaning for the dialogue planner. In making decisions about content to include in an affinity game, our current affinity game logic focuses on the relationships between and emotions associated with themes, foci, past events and information. At the planning level, the only thing the planner needs to know about the information symbols is that, when looking at two different assertions, the information symbols are the same or different. The information symbols essentially come along for the ride as the planner lays out the argument. Of course, when the conversation is actually presented to the player, some part of the system must know what these symbols mean in order to generate text. But even in the case where long symbol names are mapped by human authors into hand-written conversational lines, dynamic dialogue structure generation still provides significant authorial leverage by automatically figuring out how to combine a fixed pool of lines into many different dialogue FSMs.

Methods and Operators

In SHOP2, tasks are the fundamental building blocks of the planner. Tasks are either primitive or compound; a primitive task can be executed directly, whereas compound tasks have to be broken into smaller compound tasks or primitive tasks to be executed. Methods describe rules on how to decompose compound tasks, while operators accomplish primitive tasks. Going back to Figure 1, the higher levels of the tree

are methods and the leaves are operators. Operators are the only thing that can change the state of the world, by adding and/or deleting the predicates described in the previous section. Each method and operator takes a number of variables and checks the current bindings of them as well as the current state against a list of logical expressions called preconditions. These preconditions are what enable the planner to make intelligent choices about what to put into the plan.

```

;; start the conversation by having one person bring up a
;; focus and an emotion about it, and having the other
;; person express their own opinion
(:method (begin-conversation ?focus ?person1 ?person2 ?x)
  ;; preconditions
  (and (person ?person1)(person ?person2)
        (intensity ?x) (focus ?focus)
        (not (emotion-expressed ?person1))
        (not (emotion-expressed ?person2)))
  ;; decomposition
  ((bring-up-and-express ?person1 ?focus)
   (dismiss-and-express ?focus ?person2 ?person1)))

;; ?person1 questions what ?person2 says
(:method (counter ?person1 ?person2)
  (and (intensity ?x) (current-theme ?theme)
        (current-focus ?focus)
        (emotion-expressed ?person2)
        (or (emotion ?person2 ?focus ?emotion)
            (emotion ?person2 ?information ?emotion))
        (related-focus ?theme ?focus)
        (related-emotion ?person2 ?theme ?emotion)
        (pastevent ?pastevent)
        (will-bring-up ?pastevent ?person1)
        (information ?pastevent ?information)
        (related-pastevent ?theme ?pastevent))
  ( (!question ?person1 ?person2 ?emotion)
    (past-information ?person1 ?pastevent)
    (!!increment-intensity ?x))
  (and (intensity ?x) (current-theme ?theme)
        (current-focus ?focus)
        (emotion-expressed ?person2)
        (emotion ?person2 ?focus ?emotion)
        (related-focus ?theme ?focus)
        (related-emotion ?person2 ?theme ?emotion))
  ( (!question ?person1 ?person2 ?emotion)
    (!!increment-intensity ?x)))

```

Figure 3: SHOP2 methods for beginning a conversation and countering a statement

Consider Figure 3, which shows two methods used in creating a beat. The first method, *begin-conversation*, takes several parameters and uses the preconditions to make sure the parameters are typed correctly. These are the preconditions such as (person ?person1)(person ?person2) (intensity ?x) (focus ?focus). The rest of the preconditions check to make sure that it is, indeed, the beginning of the conversation. The predicate (emotion-expressed ?person) is added to the world state in the operator !express-emotion, so if at least one of the people in the conversation has already expressed an opinion about something, then it's not the beginning of the conversation anymore, and this is not the right

method to be calling. The method *begin-conversation* decomposes into two smaller methods, *bring-up-and-express* and *dismiss-and-express*.

The method *counter*, also in Figure 3, has a more interesting decomposition. First, this is one of three different methods that decompose the task “counter”. The other two decompose it into using past history or supporting the speaker’s own opinion; this one decomposes it into questioning the previous speaker’s opinion. Additionally, the pre-conditions do more interesting work than just variable binding. First, the planner checks to make sure that the previous speaker has expressed his/her feelings, since it’s difficult to question someone’s opinion if they have not voiced it yet. Next the planner checks to make sure that the emotion that was expressed was either about the focus of the conversation or some piece of information about a past event. This means the planner also needs to check to see if there is an appropriate past event to bring up information about. So it checks to see if there is a past event, that is both related to the current theme and that the speaker is willing to bring up, that has a piece of information that can be brought up. If there is, the planner uses the first decomposition, which both causes the speaker to question the other character’s opinion, and bring up some past history. Otherwise, if there is no past event that can be brought up, the second decomposition is chosen, which has the speaker questioning the other character’s opinion without bringing up any background.

Two of the three types of operators are represented in Figure 3, the internal operators (denoted by ‘!’) and external operators (denoted by ‘!’). Internal operators are for the use of the planner only, such as incrementing the intensity of the beat or setting the underlying story theme of the beat. The external operators become the part of the plan that will be interpreted into a finite state machine. The third type of operator (denoted by ‘!#’), is a type of operator we developed as annotations in the plan to help create a dialogue finite state machine upon interpreting the plan. In other words, these operators were used to signify conditional branches in the plan, such as planning for if the player agrees with one person versus agreeing with the other.

Planning for Finite State Machines

Since the idea is to create an interactive experience with the player, the ideal output from the planner would be a finite state machine (FSM) that illustrated how to react to different player inputs. However, non-conditional planners (such as SHOP2) generate linear plans for specific situations. Thus, we developed a way to coerce SHOP2 into planning to react differently given different player inputs. It should be noted that Kuter and Nau adapted SHOP2 to work in nondeterministic domains (Kuter and Nau 2004). In ND-SHOP2, they modified the planner itself so that it finds policies as solutions rather than plans.

Leaving the planner as it is (producing a linear plan), we initially needed to do was figure out how to take a linear plan and make it non-linear and conditional. The first step was to figure out how to take a linear plan and read it as an FSM. We started by trying to create separate plans for different outcomes, but quickly ran into the problem of how to tell

the FSM that the start node, for example, was the same in both plans.

To fix this, we created higher level tasks that basically constructed mini-plans within the overall beat plan, *plan-for-agree* and *plan-for-disagree*. The methods to decompose these tasks are very similar, since the conversation itself for if the player agrees with one character is the same as if he/she disagreed with the other character. There are different ways to decompose each task, one of which is shown in Figure 4. This method first checks to make sure the player has made a choice, makes sure that there are two different characters, and figures out which character forced the player to make a decision as this is the character that the player sided with (since this is the agree branch).

```
;; plan for agree options
(:method (plan-for-agree ?person1 ?person2)
  (and (player-response) (person ?person1) (person ?person2)
    (forced-choice ?person1)
    (different-people ?person1 ?person2))
  (!#agree-branch-start)
  (acknowledge-response ?person1 ?person2)
  (!#agree-branch-end))
  (and (player-response) (person ?person1) (person ?person2)
    (forced-choice ?person2)
    (different-people ?person1 ?person2))
  (!#agree-branch-start)
  (acknowledge-response ?person2 ?person1)
  (!#agree-branch-end) )
```

Figure 4: SHOP2 method for one possible agree branch

Rather than modifying the planner to handle conditionals, we defined a new kind of operator (‘!#’) that annotated where different branching options began and ended, as in Figure 4. The planner then creates a plan that contains both an option for the player to agree with the character who forced the choice as well as an option for the player to disagree. This allowed us to have complete finite state machines as an output from the planner that we could interpret at runtime, given the player’s input. Each state in the FSM is a semantic specification of content to express in the dialogue.

An Example Beat

Consider the situation where we, as the author, want to reveal information about a specific story theme. In the problem described above, we mentioned two story themes, the personality conflict and the family background. For this example, let us choose to reveal information about the personality conflict. To do this, we set the goal to “reveal-info” in the problem definition. The planner takes this goal and the initial state, and begins to create a plan.

First it identifies two characters to have this conversation, in our case this is Meg and Chuck, as they are currently the only two characters in the problem definition. Were the author to add more characters, the planner would pick two characters that were related to the story theme to be revealed. The planner then decomposes the goal into the task of making a conversation, which is in turn decomposed into

three smaller tasks. The first task is an exchange between the two characters, the second task is planning for if the player agrees with one character, and the third task is planning for if the player agrees with the other character.

The exchange between the two characters starts by one of them bringing up some focus of conversation. This is one way in which minor variations in the dialogue can occur, as plans are created for both Meg starting the conversation and Chuck starting the conversation. For the sake of the example, let us say Meg starts the conversation by bringing up the new video game that Chuck wants to buy and then expresses her opinion on it (she thinks it's a bad idea). Chuck dismisses her opinion by telling her she doesn't understand, then expresses his own opinion (he is excited about it).

Several exchanges of dialogue occur between the two characters before the player is forced to choose sides; it is the author's choice how long to let the conversation continue. There are three different ways of continuing the conversation: the character whose turn it is to speak can support his/her own opinion; he/she can question the other character's opinion; or he/she can bring up information about an event that happened in the past, if it relates to the current story theme. This is one of the ways that more distinct variation in dialogue occurs, as there are multiple pieces of information that could be brought up.

Once the conversation has reached the point where one of the characters forces the player to make a choice, the annotation operators ('!#') come in useful. They indicate where the cases for if the player agrees or disagrees begin and end, as well as where the end of the beat is. After the player has made a choice, there are multiple ways to end the beat. The most simple is that the character with whom the player disagreed reveals his/her belief about the story theme. There can also be a small amount of dialogue where the character tries to convince the player that he/she is right. These different options again allow for variation in dialogue.

```
Chuck says: "I can't wait til that new game comes out"
Chuck says: "I'm so excited for that new video game"
Meg says: "That's so stupid"
Meg says: "I can't believe you're getting that new video game"
Meg says: "Remember that class project?"
Meg says: "You played video games instead of putting in
the effort on our project"
Chuck says: "I don't understand why you're upset about it"
Meg says: "What do you think, player?
agree or disagree? agree
Chuck says: "Learn to have some fun in life, like me"
```

Figure 5: Example Beat

At the moment, all of the possible variations described above are found by the planner, and output to a file. This file is then read by an interpreter, which non-deterministically picks a plan to execute. The interpreter steps through the plan, at the moment doing a simple template-based generation to create a conversation as in Figure 5. The point where the player types "agree" or "disagree" is where the interpreter uses the annotations to determine which path of the finite state machine to traverse.

Related Work

Cavazza, Charles, and Mead also uses HTN planning in their interactive story-telling system (Cavazza, Charles, and Mead 2002). In their work, each character's behavior is represented by an HTN, so each character is planning to achieve an individual goal. The interesting interactions arise when the HTNs collide with one another, causing characters to interact, rather than purposefully planning that interaction. Mimesis (Young et al. 2004), another story-telling system using planning, combines a story-world plan that describes a sequence of actions for everything in the world and a discourse plan detailing cinematic effects into an integrated narrative plan. This interactive plan is then turned into a directed acyclic graph that indicates the temporal constraints in the plan. Mimesis plans at a much higher level, planning over the entire story rather than over characters as in Cavazza, Charles, and Mead's work, or over the story content in order to create dialogue as in the work presented in this paper.

At first glance, it might seem like a conditional planner would be a better choice, or better yet, an HTN conditional planner such as C-SHOP (Bouguerra and Karlsson 2004). The problem with using such a planner is that we are concerned with conditional branching that is dependent upon exogenous events, namely, the player interaction. While the current version of our planner does not include true natural language understanding, it is an area we wish to explore in the future. Therefore, we chose not to model player interaction with explicit predicates within the domain, but use the planner to allow for such events. Since most of the documented conditional planners plan over uncertain events within the planning domain, a straight conditional planner did not suit our needs. In addition, some conditional planners (Karlsson 2001) also aim to avoid the combinatorial effect of trying every possibility by pruning unpromising branches. However, our goal is to capture every possibility in a generated FSM.

Evaluation and Discussion

We have presented a prototype that procedurally generates dialogue structures for affinity games. One way to assess the viability of this approach is to characterize the generativity; that is, for a given amount of authorial effort, how many dialogue FSMs can the system generate. In our current prototype world of Meg and Chuck, there are 120 assertions (ground terms) in the content pool. Of these, 54 map directly to a dialogue output. The rest are used to make connections between assertions, enabling the planner to generate 2176 unique dialogue FSMs. In sampling the dialogue FSMs, we have found that they all make sense, and provide different slices through the content pool. Many of these variations are not "trivial" in the sense that they actually change the information being offered, though some of the FSMs represent trivial variations such as responding with a question instead of responding with information about a past event. This is a significant authorial leverage. Were the author to add a new focus, it would create twice as many FSMs as there currently are, assuming the focus was applicable to both themes. Were

the author to add a single piece of information about a single past event, this would result in approximately 16 additional possible FSMs, though the exact number may be more or less depending on how constraints associated with that piece of information (e.g. the emotion felt by a character towards that information) interacts with existing assertions.

We focus on the problem of dialogue creation, and currently use simple template methods for text realization. This work is not meant to replace the writer—the writer still needs to create the content for the planner, as well as the templated text to realize the dialogue. What we are doing is equivalent to automatically generating the dialogue trees used in many current commercial games.

Another type of planner we chose not to use is a reactive planner, since we are concerned about reacting to the player's input. Reactive planning computes the next action based on the current context, rather than creating an entire plan in advance. *Façade*'s beat selection (Mateas and Stern 2003), for example, looks at the current state and current situation, combined with some internal criteria, to choose the next beat. However, a reactive planner that doesn't do some sort of projection won't be able to create setups and guarantee payoffs. Since we actually want to be able to plan a coherent dialogue, not just react to the player input, a reactive planner is not a good choice.

Conclusion and Future Work

We have presented a planning system that automatically generates dialogue FSMs. The goal of this work is to enable richer player interaction possibilities with game characters, by enabling games to include many more, and more complex, dialogue FSMs than would be possible if all the FSMs had to be built by hand. While, in this initial work, player interaction is modeled as discrete choice (menu-based), the generated FSMs could be used with a system that employs more open ended natural language understanding (NLU), as long as the NLU system maps player input into discrete choices (discourse acts) understood by the system. While our example in the paper includes a single player choice, our infrastructure can support multiple choice points, as well as implicit choice points (where the player may interrupt or not). We are currently building a sample dialogue with multiple and implicit choice points.

We would like to connect our architecture to a more sophisticated text realizer, such as PERSONAGE (Mairesse and Walker 2007). This will allow both a more realistic conversation, as well as variation due to syntactic choices.

Within the planning domain, we intend to develop a more detailed declarative language for describing personality and emotion that can also be used by the planner when generating plans. As discussed in the introduction, there are different kinds of social games, and the factors of personality and emotional state are key in determining what social game is appropriate to start, given the state of the world. We can then begin writing task networks for additional social games.

Once we have several social games modeled as HTNs, we will create a playable NPC conversation prototype that allows us to playtest both interactive drama-style conversations, such as the affinity game, as well as the styles of dia-

logues more typically found in dialogue heavy genres such as role-playing games.

References

- Bouguerra, A., and Karlsson, L. 2004. Hierarchical task planning under uncertainty. In *Proceedings of the Third Italian Workshop on Planning and Scheduling (AI*IA)*.
- Cavazza, M.; Charles, F.; and Mead, S. J. 2002. Interacting with virtual characters in interactive storytelling. In *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Systems*.
- Karlsson, L. 2001. Conditional progressive planning under uncertainty. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence IJCAI*.
- Kuter, U., and Nau, D. 2004. Forward-chaining planning in nondeterministic domains. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI)*.
- Mairesse, F., and Walker, M. 2007. Personage: Personality generation for dialogue. In *Proceedings of the Association for Computational Linguistics*.
- Mateas, M., and Stern, A. 2003. Integrating plot, character, and natural language processing in the interactive drama *Façade*. In *Proceedings of the First International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE)*.
- Mateas, M., and Stern, A. 2005a. Build it to understand it: Ludology meets narratology in game design space. In *Proceedings of DiGRA 2005 Conference: Changing Views Worlds in Play*.
- Mateas, M., and Stern, A. 2005b. Structuring content in the *Façade* interactive drama architecture. In *Proceedings of the First International Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.
- Mateas, M. 2002. *Interactive Drama, Art, and Artificial Intelligence*. Ph.D. Dissertation, Carnegie Mellon University. Technical Report CMU-CS-02-206.
- Mateas, M. 2007. The authoring bottleneck in creating AI-based interactive stories. In *Proceedings of the AAAI 2007 Fall Symposium on Intelligent Narrative Technologies*.
- McKee, J. 1997. *Story: Substance, Structure, Style, and The Principles of Screenwriting*. HarperEntertainment.
- Nau, D.; Muoz-Avila, H.; Cao, Y.; Lotem, A.; and Mitchell, S. 2001. Total-order planning with partially ordered subtasks. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence (IJCAI)*.
- Reiter, E., and Dale, R. 2000. *Building Natural Language Generation Systems*. Cambridge University Press.
- Walters, M., and Pressey, S. 2008. Dialog production for bioware's mass effect. Lecture, 2008 Game Developers Conference. Slides at: <https://www.cmpevents.com/GD08/a.asp?option=C&V=1&SB=4&Pv=2>.
- Young, R. M.; Riedl, M. O.; Branly, M.; Jhala, A.; Martin, R.; and Saretto, C. 2004. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development*.