Proceedings, The Twelfth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE-16)

Characters Who Speak Their Minds: Dialogue Generation in *Talk of the Town*

James Ryan, Michael Mateas, and **Noah Wardrip-Fruin**
Expressive Intelligence Studio
University of California, Santa Cruz
{jor, michaelm, nwf}@soe.ucsc.edu

Abstract

The Expressive Intelligence Studio is developing a new approach to freeform conversational interaction in playable media that combines dialogue management, natural language generation (NLG), and natural language understanding. In this paper, we present our method for dialogue generation, which has been fully implemented in a game we are developing called *Talk of the Town*. Eschewing a traditional NLG pipeline, we take up a novel approach that combines human language expertise with computer generativity. Specifically, this method utilizes a tool that we have developed for authoring context-free grammars (CFGs) whose productions come packaged with explicit metadata. Instead of terminally expanding top-level symbols—the conventional way of generating from a CFG—we employ an unusual *middle-out* procedure that targets mid-level symbols and traverses the grammar by both forward chaining and backward chaining, expanding symbols conditionally by testing against the current game state. In this paper, we present our method, discuss a series of associated authoring patterns, and situate our approach against the few earlier projects in this area.

Introduction

Current authoring practice for videogame dialogue production, in which individuals or even teams of writers tirelessly produce huge amounts of content by hand, is largely seen as both untenable and constraining of the form (Mateas 2007). This issue hinders all games with dialogue, but it is most pronounced in the emerging genre that features freeform naturalistic conversation as core gameplay (Mateas and Stern 2003; Strong and Mateas 2008; Spierling 2011; Joseph 2012; Endrass et al. 2014; Brusk 2014; Evans and Short 2014; Horswill 2014; Lewis and Dill 2015; Treanor, McCoy, and Sullivan 2015; Mohov 2015; Lessard 2016), a format that has been shown to maximize player presence and engagement (Sali et al. 2010). Clearly, any alleviation of this content problem will rely on some amount of generativity, and so *natural language generation* (NLG) appears as a panacea. But NLG has yet to find its killer application in games (Horswill 2014). Elsewhere, we have argued that traditional NLG—in which heavy interlocking systems are assembled by practitioners with deep expertise—may actually be a siren song in these waters (Ryan et al. 2015a). Skilled writers who can produce stylistically rich and evocative text should be the ones writing videogame dialogue, but people who are both skilled writers and NLG experts are rare. As such, we require approaches in which authors without procedural backgrounds can still be largely responsible for the production of dialogue (as in today's untenable configuration), while somehow harnessing a nontrivial degree of computer generativity (thereby realizing tomorrow's ergonomic configuration).

In this paper, we present a new method for videogame dialogue generation that eschews a traditional NLG pipeline in favor of an approach that utilizes two complementary strengths of humans and computers—humans' deep knowledge of natural language expressivity and all its attendant nuances, and a computer's capacity to efficiently operate over probabilities and treelike control structures—while simultaneously minimizing both entities' huge deficiencies in the converse. Specifically, this method utilizes a tool that we have developed for authoring *context-free grammars* (CFGs) whose productions come packaged with explicit metadata. Instead of terminally expanding top-level symbols—the conventional way of generating from a CFG—we employ an unusual production system that works *middle-out*. That is, the system targets mid-level symbols and traverses the grammar by both forward chaining and backward chaining, expanding symbols *conditionally* by testing against the current game state. Our method has been fully implemented in a game that we are developing called *Talk of the Town*, and is part of a larger conversation engine that combines NLG with *dialogue management* and *natural language understanding* (NLU). We believe that our contribution here is a step toward generative dialogue becoming commonplace in games, and we hope that others will consider taking up our method in their own future projects.

Background

Before proceeding, we will provide background information on our target application, *Talk of the Town*, as well as Expressionist, the authoring tool that crucially underpins our approach to dialogue generation. Information on our approaches to dialogue management and NLU can be found elsewhere (Ryan, Mateas, and Wardrip-Fruin 2016b; Summerville et al. 2016; Ryan et al. 2016b).

Talk of the Town

Talk of the Town is a game built on an AI framework that generates and simulates a small American town populated by *non-player characters* (NPCs) who form and propagate subjective knowledge of the gameworld (Ryan et al. 2015b); it is currently under development. Due to space considerations, in this section we will outline only the aspects of this game that are at work in the generation of character dialogue. *Talk of the Town* simulates by day and night timesteps, and during each timestep each character will be at some location in the town. When a character is at a location, she will observe her surroundings (to build up her knowledge of the world) and may engage in social interactions with other characters at the same location (Ryan, Mateas, and Wardrip-Fruin 2016c). In one level of simulation fidelity, employed during *world generation* (Adams 2015), characters engaging in such an interaction transact in a purely symbolic exchange of information. Partially by virtue of the system we present in this paper, a higher fidelity of simulation is now supported, in which such exchanges are rendered as naturalistic character conversations with fully realized dialogue. Critically, these conversations are driven by a *dialogue manager*, which allocates speaking *turns* and enforces conversational norms. While elsewhere we describe our dialogue manager in detail (Ryan, Mateas, and Wardrip-Fruin 2016b), here we will only emphasize three of its core notions: dialogue moves, conversational obligations, and topics of conversation. As a very general notion of a *speech act*, lines of dialogue may be used to perform *dialogue moves*, *e.g.*, ‘greet’ or ‘ask about the weather’. Relatedly, lines of dialogue, when uttered, may *obligate* their recipients to perform some dialogue move in turn. For instance, the line of dialogue “How are you?” might obligate its recipient to perform the move ‘answer how are you’—we call these *conversational obligations*. When a speaker has an unresolved obligation, the dialogue manager will give her the next conversation turn, allowing her to request (from our NLG system) a line of dialogue that will perform the prescribed dialogue move. Additionally, certain lines of dialogue may be used to address existing *topics of conversation* (*e.g.*, ‘work’ or ‘the weather’) as a way of filling in lulls in the conversation. Finally, beyond resolving obligations and filling in conversation lulls, characters may pursue *conversational goals*, whose plan steps are realized by the performance of targeted dialogue moves. As such, characters begin their turns by seeking either lines of dialogue that perform certain dialogue moves (to resolve obligations or realize plan steps) or address certain topics of conversation (to fill lulls). As we explain below, these are the simple objectives that frame the operation of our NLG system.

Expressionist

Expressionist is a tool for specifying a CFG that may be used by a content generator to produce text that is explicitly annotated for the concerns of a larger application, such as a game (Ryan et al. 2016a). Expressionist’s design is meant to instantiate the *modular content* approach that we have outlined elsewhere (Ryan, Mateas, and Wardrip-Fruin 2015), which prescribes small units of content that are packaged with metadata and that may be procedurally recombined into larger units of annotated content. Using Expressionist, an author specifies *nonterminal symbols* and the *production rules* that may exhaustively expand them to produce *terminal derivations* composed fully of *terminal symbols*—*i.e.*, strings. Crucially, nonterminal symbols may be annotated using arbitrary *tagsets* and *tags* that are defined by the author; this is the core appeal of Expressionist and what separates it from Tracery, a related tool that also has CFG underpinnings (Compton, Kybartas, and Mateas 2015). When a terminal derivation is produced in a CFG, it will have expanded a set of nonterminal symbols along the way—in Expressionist, such a derivation accumulates all of the markup that an author has attributed to all of the symbols in this set. This allows an author to modularly specify capsules that contain both symbolic markup (*e.g.*, a speech act) and the rules for producing variations of the linguistic expression of that markup. Appealingly, because it is inherited from expanded symbols during the derivation process, markup does not have to be constantly reduplicated at the level of terminal symbols. In this way, the author ends up specifying a *hierarchy of markup* that is less burdensome to attribute and is easier to understand. In addition to symbol annotation, Expressionist allows production rules to be assigned probabilities, which more specifically makes the data structure authored using the tool a *probabilistic* CFG. After a session with the tool, an author exports her grammar as a JSON file that a content generator may operate over.

Dialogue Generation in *Talk of the Town*

In this section, we outline our method for dialogue generation in *Talk of the Town*. As we explain momentarily, central to this method is a module, called *Productionist*, that operates over a JSON file specifying an Expressionist grammar to produce lines of dialogue that satisfy targeted requests made by the game’s dialogue manager (on behalf of NPCs that are engaged in conversation).

Design Goals

The development of our NLG system has been driven by the following design goals:

- Allow *Talk of the Town* characters to engage in naturalistic conversation that expresses character personality and other aspects of underlying game state.
- Produce *retargetable* dialogue (Samuel et al. 2014), *i.e.*, content that can be reused across characters and contexts.
- Maximize the degree to which the *quality* of generated content depends on human authoring performance (rather than generator performance). In other words: let the human be the creative element.
- Minimize the degree to which the *amount* of generable content is limited by human authoring time. In other words: let the computer be the generative element.

System Architecture

Our NLG system architecture is illustrated in Figure 1. On a given conversation turn, the dialogue manager makes a tar-

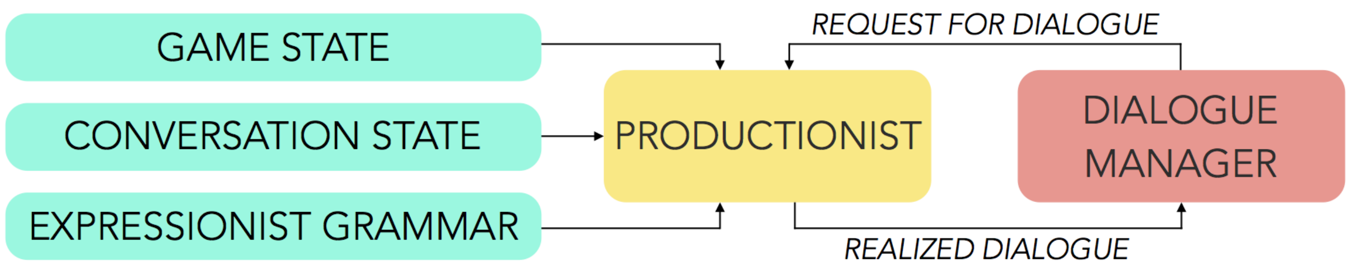

Figure 1: Our NLG architecture. The Productionist module satisfies targeted requests made by our game's dialogue manager by operating over the game state, conversation state, and an Expressionist grammar to produce fully realized dialogue.

geted request for dialogue (on behalf of an NPC) to the Productionist module, which operates over the game state, the conversation state, and our Expressionist grammar to produce a bespoke, fully realized line of dialogue.

Authored Context-Free Grammar

Using Expressionist, we have authored a grammar that is capable of generating roughly 3M unique lines of dialogue, which cumulatively may be used to perform a total of 39 dialogue moves. This grammar took approximately twenty hours for a single author to specify, and features 246 nonterminal symbols and 665 production rules. In the grammar, top-level symbols terminally expand to full lines of dialogue, whereas terminal expansions of lower-level symbols are dialogue fragments. Our tagsets, whose tags are attributed to its nonterminal symbols, correspond to the content metadata that is central to the operations of our dialogue manager (Ryan, Mateas, and Wardrip-Fruin 2016b). Specifically, each nonterminal symbol was attributed zero or more tags from the following tagsets (and a few more that we omit due to space considerations):

- **Preconditions.** Conditions specifying aspects of the game state or conversation state that must hold in order for the nonterminal symbol to be expanded during the production of a line of dialogue. We specify these as raw Python expressions that Productionist can evaluate against the game state at runtime.
- **Dialogue moves.** Dialogue moves that a line of dialogue will perform if the nonterminal symbol is expanded during the former's production.
- **Obligations pushed**. Dialogue moves that a line will obligate its recipient to perform next if the nonterminal is expanded during its production.
- **Topics addressed.** Topics of conversation that a line of dialogue will address if the symbol is expanded.
- **Propositions.** *Propositions* about the gameworld (Ryan, Mateas, and Wardrip-Fruin 2016b) that a line of dialogue will assert; these cause dialogue recipients to consider updating their beliefs about the world.
- **Lie conditions.** Conditions that, if satisfied, make a generated line of dialogue a *lie* (Ryan, Mateas, and Wardrip-Fruin 2016b).
- **Conditional effects.** Rules specifying the *conditional effects* (updates to the game state; Ryan, Mateas, and Wardrip-Fruin, 2016b) that a line of dialogue will yield.

Dialogue Requests

Productionist yields fully realized character dialogue to satisfy targeted requests made by the game's dialogue manager on behalf of NPCs. As alluded to above, such a request solicits a line of dialogue that either will perform a targeted dialogue move or address a targeted topic of conversation.

Dialogue Production

As illustrated in Figure 2, dialogue production in our method proceeds from a nonterminal symbol in our grammar with the desired markup and carries out *conditional expansion* by both *backward chaining* and *forward chaining*.

Symbol Targeting. A generated line of dialogue will perform a given dialogue move or address a given topic of conversation if a nonterminal symbol with the corresponding markup was expanded during the line's production. Given this, Productionist's first task is to collect all the nonterminal symbols in our authored Expressionist grammar that have been attributed markup corresponding to the dialogue move or topic solicited in the dialogue request. After compiling all such symbols, the module randomly shuffles them before attempting to target the first symbol in this ordering. If at any point the targeting of that symbol fails (for reasons we describe momentarily), the procedure begins from the next available symbol, and so forth. If Productionist runs out of symbols, then no line of dialogue can be furnished to satisfy the dialogue request. We note, however, that such an occurrence would represent authoring error, rather than failure on behalf of Productionist. Requests made by the dialogue manager should only correspond to generable lines, and if a line of dialogue satisfying the request is generable, Productionist will generate it. As such, any failure to satisfy a request is a cue for the human author to fix an authoring mistake or augment the Expressionist grammar.

Forward Chaining. By a process of *forward chaining*, Productionist attempts to terminally expand the nonterminal symbol that it is currently targeting. To do this, it must first verify that the symbol's preconditions are met—we call this *conditional expansion*. As noted above, nonterminal symbols in our grammar may be attributed preconditions, specified as raw snippets of Python code, that can be evaluated against game and conversation state. If a symbol's preconditions are not met on some conversation turn, it may not be expanded at that time, which means that forward chaining from that symbol fails. If a symbol's preconditions *are* met, Productionist then attempts to execute one of its production rules, by which the targeted symbol may be expanded to a set of nonterminal and terminal symbols. For example, a production rule in the Expressionist idiom looks like this: $greet \rightarrow [[greeting\ word]], [[interlocutor\ name]]$. By this rule, the nonterminal $greet$ is expanded to a concatenation of three symbols: the nonterminal $greeting\ word$, the terminal ', ', and the nonterminal $interlocutor\ name$. In order for a production rule to be successfully executed, each of the nonterminal symbols on its right-hand side (if any) must be terminally expanded (*i.e.*, must launch its own successful forward chaining). This means that the example rule could not be executed unless forward chaining from both $greeting\ word$ and $interlocutor\ name$ was successful. The process of forward chaining from a symbol is thus one of recursively executing production rules until either all available production rules have failed to execute, in which case forward chaining has failed, or a terminal expansion of the symbol has been produced. At every point of decision between a set of production rules, Productionist chooses probabilistically according to the probabilities attributed by the human author using Expressionist.

Backward Chaining. Unless it is a top-level symbol, the terminal expansion of Productionist's targeted nonterminal symbol will be a dialogue *fragment*; in such cases, we must also carry out *backward chaining* from that symbol to produce a complete line of dialogue. By this procedure, we attempt to execute a production rule that has the targeted symbol on its *right*-hand side. Here, a rule is successfully executed if all the symbols on its right-hand side can be terminally expanded (which may require forward chaining) and the symbol on its left-hand side can launch terminal backward chaining. Specifically, backward chaining terminates upon arrival at a top-level symbol whose preconditions are met. At this point, a complete line of dialogue performing the targeted dialogue move or topic of conversation will have been produced.

Why Backward Chain? Here, the reader may be wondering why the symbols associated with dialogue moves and topics of conversation are not always top-level symbols, since that would allow for the more straightforward unidirectional terminal expansion that is conventional of CFGs. The answer has to do with the nature of conversational obligations in *Talk of the Town*. In our grammar, top-level nonterminals tend to be annotated as performing coarse dialogue moves, and deeper symbols more specific moves. For instance, there is a top-level symbol associated with the coarse move *make small talk*, which has a child node corresponding to the finer move *talk about the weather*, whose own child is associated with the even finer move *respond about the weather*, and so forth for several levels. Typically, conversational obligations, which frequently inform dialogue requests made to Productionist, require a conversant to perform a finely grained move. For example, the line "How do you like this weather?" does not merely obligate its recipient to perform the move *make small talk*, but rather the more specific move *respond about the weather*. For this reason,

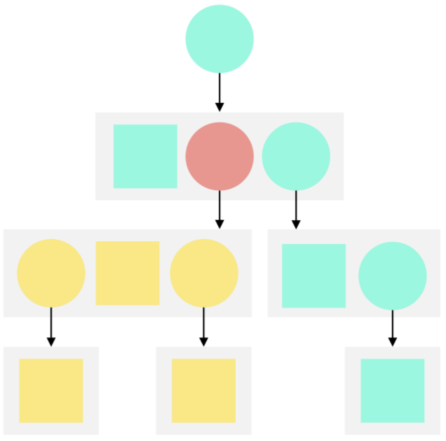
Figure 2: Dialogue production, illustrated. A nonterminal symbol with the desired markup (red) is targeted, triggering production rules (gray)—whose bodies contain terminal (square) and nonterminal symbols (circle)—by both forward chaining (yellow) and backward chaining (aqua). The resulting production (a complete line of dialogue) is the concatenation of the six terminal symbols, and comes packaged with all the markup attributed to the six nonterminal symbols.

Productionist must often target mid-level nodes, and this necessitates both forward and backward chaining.

Middle-Out Expansion. The grammar is organized in this way to support an elegant hierarchy of markup—in this case, annotations about small talk, the weather, and negative sentiment only have to be applied once each, even though hundreds of thousands of generable lines can inherit them. If all targetable nonterminal symbols were made top-level, as would be standard in CFGs, markup (including preconditions) would have to be duplicated in many places in the graph, significantly increasing authorial burden and chances for error. When combined with markup, *middle-out expansion* greatly increases the authorial leverage (and tractability) of authoring large, targetable dialogue spaces, and is a core contribution of this work.

Metadata Collection

Upon generating a line of dialogue, Productionist iterates over all the nonterminal symbols that were expanded during the line's production and collects all the markup that was attributed to those symbols during authoring. At this point, the line is packaged up with this metadata, which captures the semantic and pragmatic information corresponding to the tagsets we enumerated above.

Template Realization

Rather than a fully grounded line of dialogue, the production procedure we have outlined will have actually produced a *templated line of dialogue*, like those found in systems such as *Curveship* (Montfort 2009), *Prom Week* (McCoy and others 2014), and *Versu* (Evans and Short 2014). This is because certain pieces of a line of dialogue, such as a character's name or gendered pronoun, are not knowable at authoring time (due to retargeting), which means they must be filled in at runtime. These template gaps—called *runtime variables* in Expressionist parlance—are specified by an author using raw Python code that Productionist may evaluate (to a string) against the current game and conversation state. For instance, in the templated line "Hi, [conversation.interlocutor.name].", the runtime variable $conversation.interlocutor.name$ (enclosed in brackets) would be resolved by binding the variable $conversation$ to the Python object representing the current conversation and then evaluating the Python expression accordingly.

Deployment

Upon realizing the template, Productionist will have produced a surface-level line of dialogue that satisfies the dialogue manager's targeted request and comes packaged with metadata specifying semantic and pragmatic information about it. At this point, Productionist delivers the line to the dialogue manager, which operates over its metadata to update the conversation and game state; for lines with associated propositions, this will also invoke the recipient's *belief revision* procedures (Ryan et al. 2015b). Finally, the line is handed off to the core game engine for displaying.

Examples

While we do not have space here to include generated dialogue, examples of this, including full conversations, can be found elsewhere (Ryan, Mateas, and Wardrip-Fruin 2016b; Summerville et al. 2016).

Authoring Patterns

In this section, we discuss authoring patterns that allow our system to avoid typical NLG pitfalls and to express character personality and other aspects of underlying state.

Avoiding Repetition

As a crucial authoring pattern, we use preconditions to thwart awkward repetition of sentence structure or lexical items. For example, we may take a nonterminal symbol that expands to a specific sentence structure and attach a precondition to it that bars the symbol from being expanded if it was already used to produce an earlier line of dialogue. Likewise, symbols that expand to a salient word or phrase can be attributed preconditions that check for whether that lexical content appeared in an earlier line.

Surfacing Character Personality

There is a body of work in expressive NLG that aims to surface speaker personality in generated dialogue (Mairesse and Walker 2007; Strong and others 2007); in our method, we do this by again utilizing symbol preconditions. Specifically, we frequently employ an authoring pattern whereby certain symbols are gated according to the speaker's personality traits and expand to dialogue fragments that express such traits. As a subtle example, we often use a nonterminal called *expressive punctuation* at the end of lines; this symbol may expand to an ellipsis if the speaker is introverted, an exclamation mark if she is extroverted, or a period at any time. More broadly, we can use this general tactic to surface personality through sentence structure, lexical choice, or any other aspect of dialogue. We can even support cases where a character's personality fundamentally alters the way she performs a given dialogue move. For example, we could author symbols that expand to lines like "Get away from me." that extremely rude characters could use to perform the move *greet back*. One might worry that in such a case an NPC recipient of that line would treat it as if it were an ordinary greeting, but this is not the case: conversational obligations are asserted by individual lines (according to their metadata inherited during production), not the dialogue moves they perform, which means this line could push on obligation on its recipient to, *e.g.*, perform the dialogue move *respond to rude greeting*.

Surfacing Character Beliefs

Critically, we surface character beliefs (a core aspect of *Talk of the Town* gameplay) by a pattern that utilizes runtime variables. As noted above, runtime variables allow an author to include in terminal symbols arbitrary code that evaluates to a string at runtime. In authoring how a character might ask an interlocutor about her work life, for example, we could use runtime variables to cleverly surface multiple speaker beliefs: "So, [speaker.belief(interlocutor, 'first name')], how is the [speaker.belief(interlocutor, 'job shift')] shift at [speaker.belief(interlocutor, 'workplace')] going?" We anticipate this strategy being effective in communicating aspects of the storyworld to the player, who may learn about the town and its residents merely by eavesdropping. Elsewhere, we briefly discuss using generative character conversations for this kind of storytelling, and for *background believability* (Ryan, Mateas, and Wardrip-Fruin 2016a).

Content Sampling

Lastly, we would like to briefly note a pattern utilizing sampling techniques. First, we frequently harness Expressionist's *live feedback* feature, which allows authors to rapidly expand nonterminal symbols (or execute specific production rules) to verify the quality of generable output. Additionally, we employ a variant of *story sampling* (Samuel et al. 2014) characterized by the following authoring loop: simulate character conversations, revise or augment our grammar according to observed deficits, repeat.

Discussion and Prior Work

We plan to conduct a holistic evaluation of our larger approach to conversational interaction (which integrates dialogue management, NLG, and NLU into a gestalt) in the context of a completed playable experience. This is not yet possible, however—we are still developing our approach to NLU (Summerville et al. 2016; Ryan et al. 2016b) and other aspects of the larger gameplay experience of *Talk of the Town*. In this paper, we will focus on comparing features of our approach to existing methods for dialogue generation.

There have been a small number of projects that have taken steps toward the integration of NLG into games, but even fewer completed titles have shipped with generative dialogue. In now classical foundational work, Loyall and Bates (1997) couple embodied action and generative dialogue using an extension to the reactive-planning language Hap, while Cavazza and Charles (2005) and Rowe, Ha, and Lester (2008) select (and realize) syntactic templates by reasoning over, respectively, character affinities and archetypes. More recently, the work of Marilyn Walker and collaborators has explored the integration into games of traditional NLG pipelines proceeding from character models (Khosmood and Walker 2010; Walker et al. 2011; Lin and Walker 2011; Walker et al. 2013) as well as symbolic content representations (Lukin, Ryan, and Walker 2014; Antoun et al. 2015); these efforts are exemplified by the ambitious SpyFeet prototype (Reed and others 2011). *Bot Colony* also employs a traditional NLG pipeline—a first for a commercially released title—particularly one in the style of *service-based dialogue systems*, made possible by the narrative conceit of NPCs in the game being service robots (Joseph 2012). Even more recently, Ian Horswill's ambitious *MKULTRA* generates character dialogue from a *definite clause grammar* specified in Prolog (Horswill 2014). Dunyazad likewise takes a grammar-based approach, in its case toward generating narrational text and choice prompts (Mawhorter 2016). By employing generative grammars, these systems harness the superset power of *templated dialogue*, a pattern used in *Prom Week* (McCoy and others 2014), Versu (Evans and Short 2014), the LabLabLab trilogy (Lessard 2016), *Event[0]* (Mohov 2015), and various works of interactive fiction (Short 2014).

Like Horswill and Mawhorter, we also eschew a traditional NLG pipeline and instead utilize generative grammars. As we have argued more extensively elsewhere (Ryan et al. 2015a), we believe that traditional NLG currently incurs authorial burden (due to systemic demands) that outweighs the benefits of its generativity, is not reliable enough in terms of content quality to be used in a shipped game,[1] and is unapproachable to naive authors. When utilizing grammars, a single system has to be authored—one that can execute production rules—and thereafter the pool of generable content grows exponentially with authoring time, due to the natural combinatorics of generative grammars. This is in contrast to traditional NLG, where many interlocking systems must be wrangled and the pool of generable content tends to grow more slowly with the addition of specific realization rules. In a traditional NLG pipeline, bad outputs may emerge from a complex series of rule interactions, which makes the editing process more like debugging than authoring; in contrast, in a CFG each snippet of surface text will have been generated by a discrete production rule. Such rules are easy to correct using Expressionist's live-feedback feature, which allows for rapid checking of the results of discrete rule executions (and the expansions of individual symbols). Another major appeal of our method is that it is approachable to naive authors—Tracery continues to demonstrate the appeal of CFGs to those lacking procedural backgrounds (Compton, Kybartas, and Mateas 2015)—whereas traditional NLG pipelines demand NLG expertise.

[1]*Bot Colony* is an exception here, but one whose expectations of quality are cleverly alleviated by the narrative conceit that its NPCs are service robots.

More crucially, this is to our knowledge a novel exploration of generation by middle-out, conditional CFG expansion. While earlier approaches have operated over grammars by searching in multiple directions (Kay 1980), including for purposes of generation (Kay 1996), we do so *conditionally*, with tests being attached to candidate production rules in the style of production systems (Davis and King 1975).[2] This is critical to our approach, since it directs the bidirectional search according to authorial aims, like surfacing character personality, which may be encoded in the condition logic. This precludes the need for something like *overgenerate and rank*, which would push the specification of strategies for evaluating generable content out of the authoring interface and into the game code. That would be antithetical to our project because naive authors may not be able to write code, and more generally because we want evaluative strategies to be specified at the time of content authoring (as a form of tight authorial control for content authors).

In the future, we plan to train a team of naive authors to help augment our grammar, in similar fashion to the *Prom Week* team's inclusion of several undergraduates who were tasked with dialogue authoring (Kaltman et al. 2014). We are confident about this plan in part because Expressionist, and bespoke Productionist-like modules, have been used to generate natural language in two recently released student games—*Project Perfect Citizen* and *Snapshot*—as we have detailed elsewhere (Ryan et al. 2016a).

References

Adams, T. 2015. Simulation principles from Dwarf Fortress. In Rabin, S., ed., *Game AI Pro 2: Collected Wisdom of Game AI Professionals*.

Antoun, C.; Antoun, M.; Ryan, J. O.; Samuel, B.; Swanson, R.; and Walker, M. A. 2015. Generating natural language retellings from Prom Week play traces. *Proc. PCG*.

Brusk, J. 2014. *Steps Towards Creating Socially Competent Game Characters*. Ph.D. Dissertation, University of Gothenburg.

Cavazza, M., and Charles, F. 2005. Dialogue generation in character-based interactive storytelling. In *Proc. AIIDE*.

Compton, K.; Kybartas, B.; and Mateas, M. 2015. Tracery: An author-focused generative text tool. In *Interactive Storytelling*.

Davis, R., and King, J. 1975. An overview of production systems. Technical report, DTIC Document.

Endrass, B.; Klimmt, C.; Mehlmann, G.; Andre, E.; and Roth, C. 2014. Designing user-character dialog in interactive narratives: An exploratory experiment. *Computational Intelligence and AI in Games*.

Evans, R., and Short, E. 2014. Versu—a simulationist storytelling system. *Computational Intelligence and AI in Games*.

[2]More precisely, in our method the conditions are attached to the nonterminal symbols in the rule heads and bodies.

Horswill, I. D. 2014. Architectural issues for compositional dialog in games. In *Proc. GAMNLP*.

Joseph, E. 2012. Bot colony—a video game featuring intelligent language-based interaction with the characters. *Proc. GAMNLP*.

Kaltman, E.; Wardrip-Fruin, N.; Lowood, H.; and Caldwell, C. 2014. A unified approach to preserving cultural software objects and their development histories.

Kay, M. 1980. Algorithm schemata and data structures in syntactic processing. *Technical Report CSL80-12*.

Kay, M. 1996. Chart generation. In *Proc. Association for Computational Linguistics*.

Khosmood, F., and Walker, M. 2010. Grapevine: a gossip generation system. In *Proc. FDG*.

Lessard, J. 2016. Designing natural-language game conversations. In *Proc. DiGRA-FDG*.

Lewis, M., and Dill, K. 2015. Game AI appreciation, revisited. *Game AI Pro 2: Collected Wisdom of Game AI Professionals*.

Lin, G. I., and Walker, M. A. 2011. All the world's a stage: Learning character models from film. In *Proc. AIIDE*.

Loyall, A. B., and Bates, J. 1997. Personality-rich believable agents that use language. In *Proc. AGENTS*.

Lukin, S. M.; Ryan, J. O.; and Walker, M. A. 2014. Automating direct speech variations in stories and games. In *Proc. GAMNLP*.

Mairesse, F., and Walker, M. 2007. Personage: Personality generation for dialogue. In *Proc. ACL*.

Mateas, M., and Stern, A. 2003. Façade: An experiment in building a fully-realized interactive drama. In *Proc. GDC*.

Mateas, M. 2007. The authoring bottleneck in creating AI-based interactive stories [panel]. In *Proc. INT*.

Mawhorter, P. A. 2016. *Artificial Intelligence as a Tool for Understanding Narrative Choices*. Ph.D. Dissertation, University of California, Santa Cruz.

McCoy, J., et al. 2014. Social story worlds with Comme il Faut. *Computational Intelligence and AI in Games*.

Mohov, S. 2015. Turning a chatbot into a narrative game: Language interaction in Event[0]. In *nucl.ai*.

Montfort, N. 2009. Curveship: An interactive fiction system for interactive narrating. In *Proc. CALC*.

Reed, A. A., et al. 2011. A step towards the future of role-playing games: The SpyFeet mobile RPG project. In *Proc. AIIDE*.

Rowe, J. P.; Ha, E. Y.; and Lester, J. C. 2008. Archetype-driven character dialogue generation for interactive narrative. In *Proc. IVA*.

Ryan, J. O.; Fisher, A. M.; Owen-Milner, T.; Mateas, M.; and Wardrip-Fruin, N. 2015a. Toward natural language generation by humans. In *Proc. INT-SBG*.

Ryan, J. O.; Summerville, A.; Mateas, M.; and Wardrip-Fruin, N. 2015b. Toward characters who observe, tell, misremember, and lie. In *Proc. Experimental AI in Games*.

Ryan, J.; Seither, E.; Mateas, M.; and Wardrip-Fruin, N. 2016a. Expressionist: An authoring tool for in-game text generation. In *Interactive Storytelling*.

Ryan, J.; Summerville, A. J.; Mateas, M.; and Wardrip-Fruin, N. 2016b. Translating player dialogue into meaning representations using LSTMs. In *Proc. Intelligent Virtual Agents*.

Ryan, J. O.; Mateas, M.; and Wardrip-Fruin, N. 2015. Open design challenges for interactive emergent narrative. In *Interactive Storytelling*.

Ryan, J.; Mateas, M.; and Wardrip-Fruin, N. 2016a. Generative character conversations for background believability and storytelling. In *Proc. Social Believability in Games*.

Ryan, J.; Mateas, M.; and Wardrip-Fruin, N. 2016b. A lightweight videogame dialogue manager. In *Proc. DiGRA–FDG*.

Ryan, J.; Mateas, M.; and Wardrip-Fruin, N. 2016c. A simple method for evolving large character social networks. In *Proc. Social Believability in Games*.

Sali, S.; Wardrip-Fruin, N.; Dow, S.; Mateas, M.; Kurniawan, S.; Reed, A. A.; and Liu, R. 2010. Playing with words: from intuition to evaluation of game dialogue interfaces. In *Proc. FDG*.

Samuel, B.; McCoy, J.; Treanor, M.; Reed, A. A.; Mateas, M.; and Wardrip-Fruin, N. 2014. Introducing story sampling: Preliminary results of a new interactive narrative evaluation technique. In *Proc. FDG*.

Short, E. 2014. Procedural text generation in IF. https://emshort.wordpress.com/2014/11/18/procedural-text-generation-in-if/.

Spierling, U. 2011. Introducing interactive story creators to conversation modelling. In *Proc. Advances in Computer Entertainment Technology*.

Strong, C. R., and Mateas, M. 2008. Talking with NPCs: Towards dynamic generation of discourse structures. In *Proc. AIIDE*.

Strong, C. R., et al. 2007. Emotionally driven natural language generation for personality rich characters in interactive games. In *Proc. AIIDE*.

Summerville, A. J.; Ryan, J.; Mateas, M.; and Wardrip-Fruin, N. 2016. CFGs-2-NLU: Sequence-to-sequence learning for mapping utterances to semantics and pragmatics. Technical Report UCSC-SOE-16-11, UC Santa Cruz.

Treanor, M.; McCoy, J.; and Sullivan, A. 2015. Social play in non-player character dialog. In *Proc. INT-SBG*.

Walker, M. A.; Grant, R.; Sawyer, J.; Lin, G. I.; Wardrip-Fruin, N.; and Buell, M. 2011. Perceived or not perceived: Film character models for expressive nlg. In *Interactive Storytelling*.

Walker, M. A.; Sawyer, J.; Jimenez, C.; Rishes, E.; Lin, G. I.; Hu, Z.; Pinckard, J.; and Wardrip-Fruin, N. 2013. Using expressive language generation to increase authorial leverage. In *Proc. INT*.