Striving for Author-Friendly Procedural Dialogue Generation

Paper

Jonathan Lessard
Concordia University
Montréal, Canada
jonathan.lessard@concordia.ca

Etienne Brunelle-Leclerc
Concordia University
Montréal, Canada
mobbex@gmail.com

Timothy Gottschalk
Concordia University
Montréal, Canada
tim.gotts@gmail.com

Marc-Antoine Jetté-Léger
Concordia University
Montréal, Canada
ma_jetteleger@hotmail.com

Odile Prouveur
Université de Montréal
Montréal, Canada
odile.prouveur@gmail.com

Christopher Tan
Concordia University
Montréal, Canada
chrisvtan@gmail.com

ABSTRACT

This paper reports on an ongoing attempt to develop an author-friendly approach to procedural game dialogue generation. Various affordances of the experimental authoring tool *Expressionist* are appropriated to allow non-computer scientist authors to design virtual characters' discourse and reasoning potential. The paper describes how the *Hammurabi* game project makes use of metadata-driven context free grammars to author virtual characters that can generate not only discourse but also context-relevant decisions. The author-friendliness and generativity of the approach is discussed.

CCS CONCEPTS

•**Human-centered computing** → *Natural language interfaces;*

KEYWORDS

Dialogue systems, text generation, narrative design, context-free grammars, game design, game development

ACM Reference format:
Jonathan Lessard, Etienne Brunelle-Leclerc, Timothy Gottschalk, Marc-Antoine Jetté-Léger, Odile Prouveur, and Christopher Tan. 2017. Striving for Author-Friendly Procedural Dialogue Generation. In *Proceedings of FDG'17, Hyannis, MA, USA, August 14-17, 2017,* 6 pages.
DOI: .1145/3102071.3116219

1 INTRODUCTION

As interactive media continue to grow in richness and complexity, traditional methods of dialogue authoring for virtual characters feel increasingly stifling. The rigidity of pre-written dialogue trees–still the staple of non-playing character verbal interaction–are at odds with the dynamism and emergence of deeply simulated game worlds. In short, there are many reasons to wish for methods to procedurally generate NPC dialogue that would be relevant to emergent game-state conditions. This paper reports on our ongoing attempt to do just this for *Hammurabi*, an experimental political managements game, using *Expressionist*, a tool designed by Ryan *et al.* [6] to author metadata-driven context-free grammars. The purpose of this paper is to expose and document the appropriation of a state-of-the art albeit experimental process for dialogue generation by a design team in the context of an applied project.

Promising new techniques are not always successful in the exact terms that saw their development, but often in the unforeseen context of the design space they've opened. Who would have predicted that the popularity of ray-casting as a rendering technique would be tied to the advent of a whole new computer game genre: the first-person shooter? Although early "tech demos" and later "killer apps" are usually well documented, it is not so much the case with the in-between moments: the messy, heuristic, empirical, intuitive design process between the emergence of a new technique and the stabilization of a dominant usage. We do not know if *Expressionist* and the techniques it embodies will ever "catch on," but this is the time to document its early appropriation by designers before it is "blackboxed" and naturalized in what may eventually seem to be the "obvious" form.

In other words, this paper's contribution is not so much in the techniques presented, which are adapted from recent research, but rather in the documentation and discussion of their ongoing *appropriation* in the context of a game *design* exploration by "naive" developers (not specialists of computational linguistics). We think this can help understand the affordances of these techniques and feed back in their further development. To do this, we will first describe the project and its research context, as well as review briefly the main characteristics of *Expressionist*. We will then describe how we have a this approach to dialogue generation in the context of our own design and development pipeline partial to giving non-computer scientist game writers as much agency as possible. We will finally share observations on these design choices after a year of development.

2 A POLITICAL MANAGEMENT GAME

Hammurabi is developed in the context of a research-creation project led by the LabLabLab and investigating how characters in digital games and interactive fiction can act as "subjective interfaces" to a simulated world [4]. *Hammurabi*, started in the summer of 2016, is the project's first prototype. It was designed as minimal

FDG'17, Hyannis, MA, USA

978-1-4503-5319-9/17/08…$15.00
DOI: .1145/3102071.3116219

test scenario for the implementation of NPCs capable of expressing their views on the game's current state of affairs.

The chosen design is a reinterpretation of the early mainframe management game *Hamurabi* (sic) (Dyment 1968) (also known as *The Sumer Game*) which abstracts the government of antique Babylon to three resources: people, grain and land; and four actions: feeding people grain, sowing land with grain, buying land with grain, and selling land for grain. Random factors alter the effects of these decisions, like the varying quality of harvests and the multiple catastrophes that can plague your kingdom, its people and its resources. The game was simple enough for the code to fit on one page [1] but its wide diffusion and numerous clones are testaments to the interesting problems it offers.

The original game is framed as a conversation between *Hammurabi*, king of Babylon (the player) and his steward: "Hamurabi: I beg to report to you, in year 1, 0 people starved, 5 came to the city, population is now 100 [...] How many acres do you wish to buy?" [1] . We've decided to push further this early example of character as interface by replacing the steward with three "viziers," each with their personality and competences. Not only do they report on what's happening in the kingdom but also execute the king's orders on his behalf. The player must assign each vizier to one of three ministries: public service (feeding the population), commerce (buying/selling land) and agriculture (sowing grain); and then allocate them a budget. Whatever happens next is in the hands of the viziers: they can do exactly as they are told or decide to keep some grain for themselves.

To grain and land, we've added a third currency: public approval. When viziers' ministries do well, Hammurabi gets some "credit," but they get even more from being in closer contact to the people. They can also choose to use their own resources to fulfill their office and acquire even more popularity. This is how the management game becomes political: the king must juggle his viziers in order to make Babylon thrive while making sure none grows too popular that they could overthrow him. This is also how these characters act as a "subjective interface" to the game system: they have a perspective (and an agenda) of their own, which modulates what information they will choose to divulge to the king/player.

On each turn, viziers report on what they have accomplished with the king's money, and advise him on what he should be doing next. This is where text generation is necessary: while much simpler than most management games, *Hammurabi*'s possibility space is still too wide to manually write in advance advices and reports for all its potential game states.

3 CONTEXT-FREE GRAMMARS WITH MARKUP

Hammurabi's technical problem can be framed as a need for natural-language generation (NLG). However, the complexity of current solutions in that field come in contradiction with other design and development requirements for computer games and interactive fiction. Characters such as the viziers in *Hammurabi* need to have a recognizable "voice," their expression needs to reflect their personality. This characterization is usually rendered through the craft of game writers and narrative designers. While they often augment their scripts with some degree of computation, they cannot be expected to all acquire the level of expertise required to set-up and customize fully automated NLG systems. Even "simple" NLG systems such as *SimpleNLG* [2] can only claim that label in comparison to the complexity of competing systems.

3.1 Expressionist

Expressionist is an authoring tool designed to offer a mixed-initiative alternative to fully automated NLG and fully written content, one that:

> [...] maximally utilizes two complementary strengths of humans and computers–human's deep knowledge of natural-language expressivity [...] and a computer's capacity to efficiently operate of probabilities and large treelike control structures–while simultaneously minimizing both entities' huge deficiencies in the converse [6].

Expressionist's strategy is to "combine the raw generative power of context-free grammars (CFGs) [...] with the expressive power of free-text markup" [6]. In other words, writers can author dialogue lines in which some parts can be replaced at run-time in respect to the current game state (and recursively). For example, if it is relevant for a character to talk about what they like, but that preference is not set in advance, an author could prepare the following line:

I [[like_verb]] [[something_I_like]].

For variation, [[like_verb]] could be randomly replaced by either "like", "adore" or "enjoy". However, a random pick will not do for [[something_I_like]] as it should reflect what is true in the current context. This is where Expressionist shines in its flexibility as it allows every fragment to be arbitrarily tagged with metadata. These tags will then offer the criteria on the basis of which a specific replacement will be chosen. In CFG terms, [[something_I_like]] is a non-terminal symbol, meaning that it hasn't been rendered in text (terminal symbol) yet; it still needs to be "expanded" according to a "production rule" which could look like:

[[chicken]] [[skiing]]

The trick here is that these two fragments are marked with different tags: "LikesChicken" and "LikesSkiing" and will only be considered valid if these variables are currently set to true. If the current character happens to like chicken, the system could make a generation request to the grammar with the specification that "LikeChicken" is true, and obtain one of the following possible outputs:

I like chicken

I adore chicken

I enjoy chicken

The value of this system is the possibility of obtaining varied outputs while keeping control over what content is considered relevant in the current context. Of course, authoring these "stenciled" fragments isn't as straightforward as chaining dialogue trees (though they also can be mind-bending) but it is far from inaccessible. Also, authors have the leeway to navigate a large spectrum between very variable, recursive templates, and fully written lines which will either be selected or not according to their tags.

The output of *Expressionist* is a JSON file containing a "grammar"– a list of symbols (terminal and non-terminal), their attached tags, and their production rules. This file needs to be interpreted by some other program to generate actual text. In the terms of *Expressionist's* authors, this other module is called a "productionist" and should be tailored to the needs of the client application; especially in terms of which kind of tags should be taken into account and how they should be evaluated. *Hammurabi's* productionist is embedded in its Unity Engine project, allowing seamless integration with the other aspects of the game.

In the following sections, we will detail how we have appropriated *Expressionist's* logic for *Hammurabi* and our development pipeline. We will discuss particularly how the game frames grammars and how it handles tags.

4 GRAMMARS

4.1 Speaking grammars

In the *Talk of the Town* project, the initial application for which *Expressionist* was developed [5], a single grammar was written to generate dialogue for every character in its simulated world. This does not necessarily mean that every character talks the same way as tags can modulate variations in output in respect to game-tracked variables such as personality traits. However, this single-grammar approach poses problems in the context of larger teams. Since context-free grammars are still very uncommon in game development, there are no established workflows that would allow someone to easily understand someone else's authoring logic. The fact that almost all potential game writers are new to this concept doesn't help, of course. Also, *Expressionist* is not yet designed for multiple user collaboration which makes collective editing of a JSON file quite risky.

While *Talk of the Town* is populated with a great number of simulation-generated people, *Hammurabi*'s cast of characters is limited to a handful of advisors with set personalities. In this context, we thought relevant to think of each grammar as the voice of a single character; and attached to a single author. This means each author can "design" how their character expresses themselves exactly how they see fit, as long as everyone follows basic guidelines regarding how the grammar will be interpreted by the productionist module. This affords leeway for authors to write their character, defining how they talk and how they respond to specific game states.

As an example, here are partial report generations from three different advisors expressing their views on similar game states:

> -Long story short, a fat bunch of'em starved, pardon the pun. I'd tell you how many but… y'a know, numbers, me, that's really never gonna happen.
>
> -Your majesty, our population isn't going to make it if we keep neglecting it like this.
>
> -Feeding people is going to be difficult since locusts ate our grain. Despite the crisis, I fed them. Just not enough… the Gods are doing this because you are a prick.

4.2 Thinking grammars

In *Hammurabi*, NPCs do not only speak differently, they also think differently. Some will be more loyal, others more opportunistic, prone to stealing and lying. In *Talk of the Town*, systemic character traits and decisions are managed by the simulation and the dialogue generation acts as an expression of these underlying processes. In our perspective of giving maximum agency to game writers, we realized that the metadata-driven context-free grammars of *Expressionist* actually afford sufficient computational sophistication for authors to script their characters' decision making processes alongside their expression scripts.

In other words, the live interplay of context-free grammars' (CFGs) generativity restricted by their associated tags can lead not only to the expression of game-state relevant sentences but also to on-line decisions. For example, if a vizier is reporting on a very good harvest, that same vizier could at the same time "realize" that this is a great time to steal some grain for themselves with little chances of anyone noticing. Consequently, that vizier could actually report a moderate crop, covering for the fact that some grain was diverted along the way. This is done by using tags the other way around: not as conditions in the selection of text fragments, but as outputs–byproducts of the selection of a given fragment.

Going back to the "I like chicken" example from earlier, imagine that although we still don't know in advance whether that character likes chicken or skiing, the moment they are going mention what they like is the actual moment their preference will be settled. Using the exact same script, the "LikesChicken" and "LikesSkiing" tags will not this time be used by the productionist do discriminate between choosing one or the other. Instead, one will be chosen randomly and the associated tag will be picked up by the system to update the game state accordingly.

This doubling of grammars as both the site of expression and of decision making move us closer towards our aim of giving game writers as much agency as possible. This is in line with a lightweight data-driven approach to game development, keeping the engine code as general as possible and leaving the most flexibility in the hands of designers.

5 TAGS

The earlier chicken vs skiing examples highlight the importance and variety of roles played by tags in this approach. In order to allow *Hammurabi's* viziers to speak and think relevantly in the game's varying context, we had to define a number of different tags with different behaviors.

5.1 "Must Have" Tags

Some text fragments are written to account for specific aspects of game states and would be irrelevant if said in other contexts. For example, let's say we want the advisors to be able to report on how much grain was eaten by rats in the last year. We have three associated tags: "rats_none", "rats_low", rats_high". We could tag each of the following lines accordingly:

> **The rats left us alone this year.**
>
> **Rats were not worse than usual.**
>
> **The rats had a feast with our harvest this year!**

These tags are to be treated as "Must Have", in the sense that the request must have "rats_none" amongst its tag set for the first line to be considered a valid option. This way, we are making sure that if a vizier decides to talk about rats, they will say something true about that situation (or in the least, they will be aware of what is true if they choose to lie).

In the end, this simply amounts to a conditional: if rats_none is true, then this line is valid. It is also possible to assign tags that are in fact evaluable expressions. In this case, the tag could be: "gamevars.rats==0" for "rats_none" or "gamevars.rats¿50" for "rats_high" for example. Although this gives more granularity to define conditions for every fragment, we had initially chosen to focus on sets of predetermined tags in order to keep things as simple as possible. This afforded a clear indication to authors as to what variables characters should "care" about and also emphasized our design intention to have "fuzzier" conversation around numbers. However, the downside to the simpler on/off named tags is the quick accumulation of required predetermined tags to account for the evolution of the game system. Moving forward we intend to implement "expression evaluation tags" as well.

Hammurabi keeps track of many variables: grain stock, harvest yield, king and advisor approval, rat damage, starvation, *etc.* Not all text fragments are relevant to all variables at once. This is why text fragments need only be associated to tags relevant to their content. In the rat example, other variables such as grain stock or king approval, whatever their current value may be, will never come in contradiction to these rat-specific fragments and can be completely ignored in the process of selecting one of them.

5.2 "Nice to Have" Tags

The problem with "Must Have" tags is that they are very strict: fragments are discarded as soon as one of them doesn't match the input markup. This poses the risk of dialogue requests returning empty because no expression could be generated matching all the demands. This is not such a problem when fragments have only one tag like in the simple rat example. However, some fragments require more nuance. Advices, for example, often need to be founded on multiple conditions. If the population *and* the grain is low *and* we have a lot of land *and* the price of land high, we have a good context to advise selling land. Creating content for all the possible intersections of 4+ "must have" variables is not impossible but one would need to be entirely thorough to avoid situations of failing content requests.

Our answer to this problem is to consider some tags to be simply "Nice to Have", meaning that their presence makes the current fragment more relevant, but their absence doesn't make it irrelevant. Technically, this amounts to *ranking* fragments in terms of their relevance. In the previous advice example, "kingdomland_high", "population_low", "landprice_high" and "kingdomgrain_low" can all be set as "Nice to Have" tags. The more they will be present, the better this advice will be, and the more likely it is to be actually said. However, it could still be chosen with one or two of the variables not matching for a lack of better available alternatives. The author's responsibility is then to phrase the output in such a way that it doesn't assume all these to be true when the character will talk. The vizier could say: "I think the stars are aligned for selling some of our land in preparation for future challenges."

The "Nice to Have" tags are also used to add nuance and flavor in respect to advisor-related variables such as their current mood, their opinion of the king or whether they are occupying their favored ministry.

5.3 Output Tags

As discussed earlier in the context of embedding decision-making into the dialogue generating grammars, tags can go both ways. They can feed into the grammar as game state data to inform the selection of text fragments; but they can also be fed back to the system as data on what the grammar chose to say and/or do. We call these "Output Tags".

Hammurabi's main use of output tags is to record a vizier's decision concerning their opportunity to secretly tamper with budgets– that is either stealing or using their own treasury to supplement insufficient funding in order to reap public approval. Since context-free grammars essentially act as a machine that assembles text fragments, the output of the "thinking grammar" is an empty expression (no text) accompanied by a number of output tags that can look like:

steal; steal; do_nothing; give; do_nothing; steal

As the productionist was traversing the grammar, selecting fragments according to their "Must Have" and "Nice to Have" tags, it also collected the symbol's "Output Tags". This is akin to someone internally weighing the pros and cons of different courses of actions in the light of various variables. The game system then tallies the tags and implements the most represented decision: steal!

Other output tags are used to change character-related variables such as their mood, their opinion of the king and of the ministry they are currently holding. Output tags are also part of the character memory system which is the object of the next section.

6 CHARACTER MEMORY

One of the advantages of dialogue-tree systems is that the current state of the conversation is never ambiguous. For every line, the author usually has a clear idea of what was said earlier. The downside of this is that any nonlinear navigation of the conversation needs to be scripted manually. In comparison, context-free grammar driven conversations are practically stateless; causing NPCs to seem stuck in an eternal present as is often the case with chatbots as well [3]. Specific strategies need to be devised to ensure some form of memory or progression.

For *Hammurabi*, we wanted the characters to be able to change over time and appear to remember what happened in the previous turns: develop competence in a ministry, for example, or, on the contrary become aggravated by being maintained many years in an unwanted office. Once again, we wanted to leave this potential in the hand of authors. And once again, this was implemented through the use of tags.

6.1 Custom Variables

If an author wants to develop a "mini-story" over multiple turns, they need a way to record that a specific thing was said so that their future selves can pick up on that information and push the narrative

further. Imagine a vizier in the agriculture ministry having a strong hunch:

> **The omens promise a bountiful harvest this year, all the grain invested in agriculture will come back tenfold!**

The author tags this fragment with "Memory_GoodOmen". As a result, the system creates a new variable GoodOmen set at value 1 and all subsequent dialogue requests are accompanied by the tag GoodOmen_1. This allows the preparation of two other fragments following up on whether the prediction turned out to be true or not. The first one is tagged with "GoodOmen_1" and "harvest_high" as "Must Haves", and the other with "GoodOmen_1" and "harvest_low":

> **As you can see, I'm truly the expert in agricultural matters**

> **So... I fired the stupid augur who misled us last year**

From there, the author could choose to start two other threads with other custom vars such as AgriculturalExpert or AgriculturalInept. A thread can also be further incremented to be developed over more than 2 turns.

7 DISCUSSION

It is difficult at this stage to draw conclusions on the use of metadata-driven context-free grammars for author-friendly game dialogue generation. *Hammurabi*'s development is still not over (planned for Fall 2017), its technology is new and still in development, its game design is new and hasn't been assessed through play testing, and all the people working on the project are using these concepts and authoring techniques for the first time. In other words, *Hammurabi* is breaking new ground on all fronts and it is hard to distinguish structural issues inherent to the chosen approach from implementation and design issues linked to inexperience with the approach or lack of established design patterns. However, we can still begin to discern some interesting features, issues and future development avenues.

7.1 Author-Friendliness

In the current state of things, we doubt most game writers would qualify *Expressionist* and our general character dialogue scripting approach as particularly author-friendly. With already many months of experience, our own authors are still sometimes puzzled by the peculiar logic of context-free grammars and necessity to juggle different flavors of game tags. Considering the novelty of this whole process, it is not even clear what "author-friendly" precisely means in this context. Our current heuristic is that the tools should afford authors to (relatively) painlessly: (1) enter new content; (2) predict runtime behavior; (3) identify, and (4) fix errors. *Expressionist*'s current interface makes the process of entering and fixing content (#1 and #4) relatively smooth. The main difficulty for authors is that the mechanisms of text generation (the process of crawling through a grammar and choosing symbols to expand) is not exposed which makes it difficult to conceive a clear mental model of what is happening (#2) and eventually troubleshoot errors (#3).

What definitely helped alleviate this was providing authors with a thorough (albeit crude) visualization of the process leading to each production:

```
******** Expanding ChrisAdvice
Adding [[AGR]] 6 times
chosen: [[AGR]]
  expanding AGR
  Adding [[OPEN]]. [[AGR_ADVICE]]. [[CLOSE]]. 1 times
  Adding [[OPEN]]. [[AGR_STATE]]. [[CLOSE]]. 2 times
  chosen: [[OPEN]]. [[AGR_STATE]]. [[CLOSE]].
    expanding [[OPEN]]
    Adding [[OPEN_OPINION_OFFICE]] 6 times
    Adding [[OPEN_OPINION_KING]] 6 times
    chosen: [[OPEN_OPINION_OFFICE]]
      expanding OPEN_OPINION_OFFICE
        Adding [[Greet_]] [[King]], [[OpenComment_Neutral]] 1 times
        chosen: [[Greet_]]
          expanding Greet_
          Adding Hail 1 times
          Adding I greet thee 1 times
          chosen: Well met
...
```

Seeing this helped authors refine their understanding of context-free grammars to both better predict behaviors and troubleshoot errors. Embedded visualization tools in *Expressionist* (ideally aware of game-state context) would certainly increase authors' efficiency.

Though our "one character, one grammar, one author approach" was chosen to allow writers to freely design their character's internal logic, the team ended up drafting quite elaborate grammar templates to make sure every character covered all requirements of the game in terms of expression and reasoning. Perhaps this would be different with seasoned authors. However, standardized templates might also be the way to go, especially to get started. Nothing would then stop an author to modify the given structure or expand it once they've understood what it is doing.

7.2 Generativity and Modularity

One of context-free grammars' main promise is that of generativity, in the general sense of being able to produce more possible outputs than they explicitly contain in their definition. Indeed, it is easy to write grammars that have the potential to yield astronomical numbers of different generations. It is one of *Expressionist*'s arguments that in comparison with traditional authoring techniques, CFGs can generate much more content for much less human labor [6]. We have not until now been able to really make good use of this feature. That is not to say that we haven't been able to write grammars with high combinatorial potential, but mostly that these variations didn't feel meaningful. It is relatively simple to have many sections of sentences vary:

[[greeting]], [[howareyou]]?[[mynameis]] Joe.

Let's say [[greeting]] can expand into "Hi", "Hello", "Howdy", "Yo", "Good morning"; [[howareyou]] into "wazzup", "how are you", "how are you doing", "howzzit"; [[mynameis]] into "my friends call me", "my name is", "I'm", *etc.* This simple production rule can indeed expand into a number of unique combinations. However, in our attempts, this form of variation felt in the end quite shallow. We would generally remember the salient content of a report–the

harvest was good–and if it were to come up again, even if the surface phrasing would change, it would still feel repetitive. What did feel different was when we authored higher-level variations such as: "we're crumbling under a ton of grain, it's raining bushels!" and "the peasants worked like machines, the harvest is crazy good!". Technically, this means that our CFGs vary at the higher level of phrases rather than words. A typical template will look like:

[[greeting]] [[mood_comment]] [[agriculture_report]] [[performance_comment]] [[goodbye]]

Each of these non-terminal symbol further splits in smaller fragments but rarely reaching the granularity of words to achieve generativity at the level of actual formulation.

This is not to say that the *Expressionist* approach isn't working. On the contrary, it did offer a solution to our original problem of dialogue generation. However, the efficiency is not so much at the level of *generativity* rather than that of *modularity*. In other words, we have in fact set up a procedural textual assembly chain. In their paper on the design challenges of interactive emergent narrative [7], Ryan *et al.* highlight the need to elaborate strategies that will allow textual representation of algorithmic processes. One of their suggestions is to borrow an approach that is very common in digital audio and visual design: "compositional representational strategies" based on content modularity, *i.e.* the breaking up of content in small elements that can be recombined to express a wide variety of meaningful compositions. In games, this takes the form of repeatable textures, reusable level-design components, graphical 3D building blocks, *etc.* Music is also very familiar with the sequencing of sometimes very small units: samples. This type of reasoning has not really made its way to interactive writers who still seem to hold the sentence as the smallest unit of expression, and hypertext linking the main mechanism of assemblage.

Using *Expressionist* and context-free grammars forced us to think our character's discourse potential as *fragments* to be assembled according to varying game states. This is how we can manage to have characters speak relevantly in a wide possibility space without having to cater for every single intersection of specific variables: each fragment cares for its own variables and are then assembled in a string of fragments that are all relevant in their own respects. In this light, *Expressionist*'s contribution might not be so much a technical one (the underlying technology, is after all, quite simple) but an invitation for authors to embrace a different authorial paradigm.

Writing textual fragments at a phrase level in order for them to be suitable for procedural composition will require a new form of authorial artistry, a body of writing techniques, just like designing repeatable textures or fabric patterns require specific methods different from free canvas illustration. How to make a text fragment "seamless," suitable for varying contexts? What is the good size of a fragment? How to think its "borders?" *Hammurabi*'s approach to dialogue generation addresses not only technical and game design problems, but also a new body of interactive writing questions.

7.3 Tag Flow and Actual Dialogue

Acute readers will raise the objection that although this paper frames its problem as one of dialogue generation, the game *Hammurabi* as described seems to only stage monologues–the viziers are reporting to the king without the latter ever answering (other than implicitly). Actual dialogue between NPCs themselves and with the player is part of LabLabLab's future research. A design intuition raised by working with the Expressionist approach is that two characters can establish a "tag flow" circuit. In other words, the output tags collected from the text generation of one character can become the input tag for their interlocutor's text generation, and so on.

This has already been successfully implemented in another prototype in development currently titled *Les Amours de Jacques*. Amongst promising features is the possibility to either let the grammars generate a dialogue automatically, or position the player as any of the interlocutors. To implement this interactive mode, we have the grammars generate a number of possible lines for the player to choose from in the form of a traditional dialogue menu.

8 CONCLUSION

Expressionist's approach to context-free grammars opens a promising design space for games and interactives fictions. It prompts authors and designers to embrace a modular approach to text in order to create virtual characters that can express themselves dynamically. However, this will remain a theoretical potential as long as actual works don't appropriate it convincingly. *Hammurabi* currently manages to generate monologues for its characters but has yet to demonstrate the aesthetic value of this feature. Now that we have a technique, our research efforts must shift to the domain of design and authorship.

ACKNOWLEDGMENTS

This research was made possible by the *Research-Creation Support Fund* of Québec's Fonds de Recherche – Société et Culture (FRQSC).

REFERENCES

[1] David H. Ahl. Hammurabi. In *Basic Computer Games*, David H. Ahl (Ed.). Microcomputer Edition, 78–79.

[2] Albert Gatt and Ehud Reiter. SimpleNLG: A Realisation Engine for Practical Applications. In *Proceedings of the 12th European Workshop on Natural Language Generation* (2009) *(ENLG '09)*. Association for Computational Linguistics, 90–93.

[3] Jonathan Lessard. Designing Natural-Language Game Conversations. In *1st International Joint Conference of DiGRA and FDG* (2016).

[4] Jonathan Lessard and Dominic Arsenault. The Character as Subjective Interface. In *ICIDS 2016, Los Angeles, CA, USA, November 15-18, 2016, Proceedings 9* (2016). Springer, 317–324. http://link.springer.com/chapter/10.1007/978-3-319-48279-8_28

[5] James Ryan, Michael Mateas, and Noah Wardrip-Fruin. Characters Who Speak Their Minds: Dialogue Generation in Talk of the Town. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference* (2016-09-19). https://www.aaai.org/ocs/index.php/AIIDE/AIIDE16/paper/view/13997

[6] James Ryan, Ethan Seither, Michael Mateas, and Noah Wardrip-Fruin. Expressionist: An Authoring Tool for In-Game Text Generation. In *Interactive Storytelling* (2016-11-15). Springer, Cham, 221–233. https://link.springer.com/chapter/10.1007/978-3-319-48279-8_20

[7] James Owen Ryan, Michael Mateas, and Noah Wardrip-Fruin. Open Design Challenges for Interactive Emergent Narrative. In *Interactive Storytelling*, Henrik Schoenau-Fog, Luis Emilio Bruni, Sandy Louchart, and Sarune Baceviciute (Eds.). Number 9445 in Lecture Notes in Computer Science. Springer, 14–26. http://link.springer.com/chapter/10.1007/978-3-319-27036-4_2