What's the Worst Thing You've Ever Done at a Conference? Operationalizing *Dread*'s Questionnaire Mechanic

Ian Horswill and Ethan Robison

EECS Department, Northwestern University, 2133 Sheridan Road, Evanston IL
ian@northwestern.edu, yikes.gov@u.northwestern.edu

Abstract

Dread (Barmore et al. 2005) is an award-winning tabletop horror RPG that emphasizes storytelling. One of its innovative mechanics is the use of a questionnaire for character design that helps players develop a personality and history for their characters. Questionnaires vary for different scenarios and characters, but always include "intrusive" questions such as "why are you married if you aren't in love?" or "what's the worst thing you've ever done to a loved one?" In this paper, we discuss work in progress on *AutoDread*, an implementation of *Dread*-style questionnaires for video games. The system presents human-authored questions and candidate answers to players to choose between, collecting the implications of those answers in a character model. As implications are accumulated, the system uses a SAT solver to filter questions and answers that are inconsistent with character facts established by previous questions.

Introduction

Dread (Barmore et al. 2005) is a table-top horror RPG in the recent tradition of "storygames" or "freeform" games, such as *Fiasco* (Morningstar 2009) and *Monsterhearts* (Alder 2012), that deemphasize rules and stats in favor of freeform improvisation (Stark 2014). It won the 2006 Ennie award for Innovation, as well as being nominated in both the Best Rules and Best Game categories.

Dread introduces two novel mechanics. One is the use of a Jenga tower in lieu of dice to determine outcomes of character actions. This provides a particularly visceral implementation of LeBlanc's aphorism that drama = uncertainty + inevitability (LeBlanc 2005).

Dread's other novel mechanic is the use of questionnaires for character design. Scenarios are pre-authored by a host (gamemaster) to have a designated set of character roles, such as *ship's captain* or *camp counselor*. Each character role has a pre-authored questionnaire that is answered, in character, by that character's player before the start of the game. The questionnaire helps the player design their character, establishing useful bits of backstory that can be used as narrative hooks by both player and host to produce drama and tension during gameplay.

Questionnaires cover a range of topics: the character's motivations, capabilities and limitations, relation to the overall plot, and relationships with other characters. Questionnaires always include so-called "intrusive" questions: questions that get at issues characters would be reluctant to disclose in real life, such as "who else knows you're a fraud?"

In this paper, we discuss work in progress on *AutoDread*, an electronic version of *Dread*-style questionnaires, suitable for use in video games. The system works from a stock of human-authored questions and potential answers, each tagged with their logical implications. For example, a question such as "How long has it been since you saw your brother?" would be tagged with "has brother". The answer "Not since mom's funeral" would be tagged "mother dead", while the answer "We watch the game every Monday night" might be tagged "brother alive" and "loves brother", or at least "not hates brother".

This brings up one of the fundamental limitations of *AutoDread*: it can't accept freeform answers from players, requiring them instead to select from a specific list of human-authored answers to the question. This allows the questionnaire to be machine-readable, at the cost of increased work for the author and decreased player freedom.

As the system asks the player questions, it accumulates these implications, as well as other facts that follow from the implications, to gradually create a model of the character and their backstory. The system uses a SAT solver (Horswill 2018) to automatically reject questions or answers that contradict facts already established in previous questions and player-selected answers. At the end, it uses the

SAT solver to generate a specific, random, character model consistent with the player's answers.

Potential applications

AutoDread is an exploratory first-step toward making a generative interactive fiction system mimicking at least some of the player experience of *Dread*. To be useful for an electronic game, the game would have to have sufficient generativity to make use of the formal character model. While it might conceivably increase player engagement for them to know their character had had a pet rabbit as a child, it won't affect the gameplay unless the game's AI can somehow take advantage of that knowledge.

Alternatively, the system could be used as an adjunct to existing tabletop games, either to help beginning players design their characters, or more likely to help GMs quickly flesh out random NPCs that they need. The GM could use generic questionnaires for merchants, innkeepers, *etc*., when a given character types was needed. Or, since the system is driven by a SAT solver, the system could generate answers to the questions itself and simply present the finished character model to the GM.

More generally, the notion of using a questionnaire as a kind of user-interface to allow a player or GM to drive a PCG system might find applications in other areas.

Questionnaire design

Questions, answers, and their implications are specified in a human-authored text file. Questions are prefixed with `Q:` and answers with `A:`. Implications for a question or answer, if any, are given in a comma-separated list beneath their respective question or answer. Questions and answers are free-form text. Implications are given as atomic propositions (single tokens) or applications of predicates to terms, expressed in ersatz English. For example:

```
Q: What could you have done to save your
brother's life?
   dead brother, family_guilt
A: Taken the car keys away
   alcoholic brother
A: Taken him away from dad
   abusive father
A: Made him move in with me to get him
out of the neighborhood
   brother_gang_member
```

Single-token facts such as `family_guilt` are atomic propositions and multiword constructions such as `dead brother` are predicate applications (i.e. dead(brother)). Possible English surface realizations of predicates are specified as part of the predicate.

The act of asking the question implies that the character's brother is indeed dead, and that the character feels some responsibility for it. It therefore forecloses any questions that presuppose the brother is still alive or that the character is an only child. The first answer, if chosen, further adds that the character's brother was an alcoholic, while the second adds that their father was abusive. The third answer adds that the brother was a gang member.

The questionnaire can also specify a limited set of constraints and other axioms:

- `Mutually exclusive:` $fact_1, \ldots, fact_n$
  At most one of $fact_1, \ldots, fact_n$ can be true.
- `Contradiction:` $fact_1, \ldots, fact_n$
  $fact_1, \ldots, fact_n$ cannot all be simultaneously true.
- `Unique:` $fact_1, \ldots, fact_n$
  Exactly one of $fact_1, \ldots, fact_n$ must be true.
- *conclusion* `<-` $premise_1, \ldots, premise_n$
  Classical implication.
- *conclusion* `<=` $premise_1, \ldots, premise_n$
  A Horn rule with stable-model semantics (Gelfond & Lifschitz 1988; Gebser et al. 2012). This differs from classical implication in that it adds the constraint that the *conclusion* may only be true if one of its Horn rules justifies it.

These are used to specify, for example, that a character must have an "affliction" – some character flaw or other disability, while preventing the character from being riddled with multiple afflictions:

```
// Afflictions
Unique: insomnia, violent, asthma,
grief_stricken, ignored, tone_deaf,
bored, superstitious, bad_temper
```

Knowledge representation language

The questionnaire is a more author-friendly front-end to the internal KR language used by the character modeling system. Character models are represented in an order-sorted logic with atomic terms: terms (arguments to predicates, represented internally by strings) are atomic rather arbitrary term expressions; they're divided into sorts (data types); and those sorts are ordered (types can have subtypes) with Entity being the top sort. Predicates specify sorts for their arguments: dead(x) is limited to x's of the Person sort.

One of the goals of the system is to allow non-technical authors to write questions and answers for the system. As a result, we've been reluctant to add rules with explicit variables and quantifiers, such as would be found in Prolog

(Clocksin & Mellish 2003) or ASP (Smith & Mateas 2011) to the system. Instead, we've added two simple higher-order constructs that handle the limited use-cases that have come up so far.

The first of these is existential quantification over a sort. If the author writes "dead father", i.e. $\mathrm{dead}(\mathrm{father})$, that has the usual semantics since father is a term. But if they write "dead parent", since parent is a sort, it means $\exists x \in \mathrm{parent}.\,\mathrm{dead}(x)$. A question or answer can therefore add to the character model that they have a dead parent, without having to specify which parent. The current language, however, does not have a syntax for specifying that all parents are dead (apart from saying "not living parent" or explicit quantified variables such as $\exists x.\,\mathrm{dead}(x) \wedge \mathrm{abusive}(x)$, i.e. they have a dead, abusive parent.

The other form of implicit quantification is the ability to specify that one predicate generalizes another predicate. There are three forms of generalization:

- p generalizes q
  $\forall x \in s.\,q(x) \rightarrow p(x)$, where s is the sort over which q is defined.
- p strongly generalizes q
  $\forall x \in s.\,q(x) \rightarrow p(x)$, but also, $p(x)$ implies at least one of the predicates that is strongly generalized by it must be true of x.
- p negatively generalizes q
  Asserts that $\forall x \in s.\,q(x) \rightarrow \neg p(x)$.

For example, `loves` and `hates` are each generalized by `exists`. So, if you love or hate someone, they must exist. But `living` and `dead` are *strongly* generalized by `exists`, so if someone exists, they must also be living or dead. `Living` and `dead` also negatively generalize one another, so you can't be both living and dead.

To simplify the English parsing and generation, predicates are currently limited to at most two arguments.

We do not presently have an author-friendly syntax for specifying new predicates or sorts inside the questionnaire file. They're currently defined in C# code.

Implementation

AutoDread is implemented in C# under the Unity3D (Unity Technologies 2004) game engine. It uses the CatSAT logic programming system for back-end inference. The KR language is translated at run-time into CatSAT assertions and character models are computed by solving for models of the assertions.

Axiom compilation

Before the questionnaire can be administered, the information in the questionnaire must be compiled into propositions and axioms in the SAT problem. Each potential fact in the questionnaire, be it an atomic fact such as "insomnia", or a predicate instance, such as "likes alcohol" is mapped to a proposition in the SAT problem. The system also generates implication rules for any generalizations or existential quantifications over the predicate instances.

Let p be a unary predicate defined over sort S, and let $D \subseteq S$ be the set of arguments to p that appear as facts in the questionnaire. Then, to implement generalization, for each $a \in D$, the system adds the rule:

$$q(a) \texttt{ <- } p(a)$$

For each negative generalization n and each $a \in D$ it adds the rule:

$$\neg n(a) \texttt{ <- } p(a)$$

And for each strong generalization Q and each $a \in D$, it adds the rule:

$$Q(a) \texttt{ <= } p(a)$$

To implement existential quantification over sorts, for each subsort $S' \subseteq S$, and for each $a \in S' \cap D$, the system adds the rule:

$$p(S') \texttt{ <= } p(a)$$

CatSAT operates internally on clauses in conjunctive normal form, i.e. disjunctions of literals (propositions or their negations). The system translates standard `<-` implications into single CNF clauses, i.e. $q(a)$ `<-` $p(a)$ is translated into the single clause $q(a) \vee \neg p(a)$. However, `<=` rules are more complicated, since they have the semantics that the consequent can only be true if the antecedent of one of its `<=` rules is true. A set of rules, such as:

$$q(a) \texttt{ <= } p(a)$$
$$q(a) \texttt{ <= } r(a)$$

is translated first into the biconditional $q(a) \leftrightarrow p(a) \vee r(a)$, which is then translated into its CNF form:

$$q(a) \vee \neg p(a)$$
$$q(a) \vee \neg r(a)$$
$$\neg q(a) \vee p(a) \vee r(a)$$

Constraint directives such as `unique:` and `mutually exclusive:` are directly supported by CatSAT, and so

require no preprocessing other than to map facts in the questionnaire to the internal SAT propositions used to represent them.

Questionnaire administration

The core loop steps through questions, presenting them and collecting answers. Questions and answers inconsistent with the current model are eliminated without being presented to the user.

The valid answers $V(q, F)$ to a question q, given a set of assumed facts F is the subset of q's answers $A(q)$, for which the assumptions F, q's implications $I(a)$, and that answer's implications are all consistent, i.e., they have a model:

$$V(q, F) = \{a \in A(q) : \exists M. M \vDash F \cup I(q) \cup I(a)\}$$

Here the model M is found by invoking the SAT solver on $F \cup I(q) \cup I(a)$.

The basic loop is then to accumulate a set of facts F by asking questions that don't contradict F, adding any implications of those questions and answers to F:

```
foreach q ∈ questionnaire {
  // Find the non-contradictory answers
  A = V(q, F)
  // Make sure there are enough of them
  if |A| > 1 {
    present q and A to the user
    collect user's answer a ∈ A
    // Update the known facts about the character
    F = F ∪ I(q) ∪ I(a)
  }
}
Find a character model M such that M ⊨ F
Display M for user
```

The algorithm is fully implemented and running in Unity.

Annotated example run

To get a sense of the operation of the system, we give an example run using a minimal questionnaire, which can be found in the appendix. "Implications" gives the set of specific implications of the questions and answers (the set F from above). "Also inferred" gives other propositions that the SAT solver would determine to be true, but that don't appear in the set F:

Q: What'll you have to drink?
A: Whiskey, on the rocks.
Implications: likes alcohol, simple tastes
Also inferred: not posh tastes, since that contradicts simple tastes.

Q: When you have trouble sleeping, what do you focus on?
A: The last time my brother and I saw each other.
Implications: likes alcohol, simple tastes, insomnia, brother exists, sentimental
Also inferred: since the character is inflicted with insomnia and they aren't allowed multiple afflictions, the other inflictions have been ruled out. Since they have been established as sentimental, other mindsets such as being arrogant have also been ruled out (this is not a realistic personality model, but it's one of the axioms).

Q: What could you have done to save your brother?
A: Take him away from Dad.
Implications: likes alcohol, simple tastes, insomnia, brother exists, sentimental, dead brother, dead brother, family_guilt, abusive father
Also inferred: hates father, not brother living, guilt (from family_guilt)

Q: What book do you read every year on the anniversary of your father's death?
A: King Lear
Implications: likes alcohol, simple tastes, insomnia, brother exists, sentimental, dead brother, dead brother, family_guilt, abusive father, narcissistic father
Also inferred: dead father. Note that an option involving the father's favorite book is suppressed here because the character hates the father.

Q: Why are you the black sheep of the family?
A: I'm a fucking loser
Implications: likes alcohol, simple tastes, insomnia, brother exists, sentimental, dead brother, dead brother, family_guilt, abusive father, narcissistic father, black_sheep, hates self

Q: How often do you see your family?
A: Never
Implications: likes alcohol, simple tastes, insomnia, brother exists, sentimental, dead brother, dead brother, family_guilt, abusive father, narcissistic father, black_sheep, hates self, living family
Also inferred: since there are living family, but the brother and father are dead, it knows there must be another living family member

Finally, the SAT solver finds a model of the implications. This fills in the gaps left unspecified by the player's answers to make a random character model consistent with the inferences: the character has a mother, whom the character

neither loves nor hates; they have a pet; they're gregarious, frail, and religious.

Related work

We are not aware of any previous work on character PCG using questionnaires. However, there is a sizable body of work on SAT-based PCG in general using Answer-Set Programming (Smith et al. 2012; Smith 2011; Nelson & Smith 2016). There have also been several attempts to generate or otherwise model character personality and history. *The Sims 3* (Maxis 2009) used a rule-based system and a set of 81 different personality traits to model character behavior (Evans 2009). Current versions of *Dwarf Fortress* (Adams & Adams 2006) use a similar system. *Versu* (Evans & Short 2013) uses a general, declarative logic for character modeling.

There has also been a significant amount of work on making author-friendly languages for interactive fiction. Ingold (2015), co-designer of *Ink* (Inkle 2013), has argued persuasively for the importance of IF scripting languages that allow writers to write their lines without having to learn programming. Nelson's *Inform 7* (Nelson 2006a) is the most widely used parser-based IF authoring system in the world. Nelson argues that within the IF domain, English can be used effectively as a declarative language (Nelson 2006b). He also built a system, *Prompter* (Nelson 2013) to act as a more author-friendly front-end to *Versu*.

Conclusion

AutoDread is a work in progress. At this point, we have a working parser, inference system, and driver loop. The next step is to build out a more substantive questionnaire, which we'd like to do in conjunction with writers. Our hope is to build a system that technophilic non-programmers can author for. Such users are often found in game development, IF authoring, and table-top roleplaying.

This will certainly involve extending the system. As mentioned above, adding new predicates (and hence new verbs) currently requires editing the underlying C# code, in part because it requires giving the parser hints as to how to translate between the internal form of the assertions and the pseudo-English of the questionnaire. It would also be desirable to find natural English expressions for the higher-order assertions about predicates, such as generalization. However, it's unfair, or at least unrealistic, to ask naïve users to learn the difference between the material implication of classical logic and Horn clauses with stable-model semantics, both of which are supported in the system.

It may also be necessary to extend the expressiveness of the system's KR language. However, what types of extensions are necessary are best determined by putting the system in front of users.

Appendix: questionnaire used in the example

```
guilt <- family_guilt

// Personalities
Unique: gregarious, playful

// Mindsets - every character has one
Unique: nostalgic, arrogant, peaceful,
optimistic, sentimental, prepared, ob-
noxious, vulnerable, bossy, health_ori-
ented

// Afflictions
Unique: insomnia, violent, asthma,
grief_stricken, ignored, tone_deaf,
bored, superstitious, bad_temper

// Body types
Unique: athletic, frail

Mutually exclusive: simple tastes, posh
tastes

// QUESTIONS
Q: What'll you have to drink?
A: Whiskey, on the rocks.
   likes alcohol, simple tastes
A: A bottle of spring water, please
   posh tastes
A: A diet coke.  I'm trying to watch my
weight.
   health_oriented, likes sweets
A: An ice-cold bottle of orange juice.
I'm parched
   likes sweets

Q: When you have trouble sleeping, what
do you focus on?
   insomnia
A: The last time my brother and I saw
each other
   brother, sentimental
A: The time I won a big game back in
high school
   athletic, nostalgic
A: What I'll say to my lover when I
make it back
   optimistic, lover
A: The serenity of mountain climbing
```

```
   likes outdoors, peaceful, athletic


Q: What do you have in your pockets?
A: My inhaler.  I'm not in the best of
health.
   asthma, frail
A: My trusty-dusty pocket knife. You
never know when something (or someone)
will need cutting.
   prepared, violent
A: A battered paperback novel. Rule
number two: always have something to
read.
   prepared, likes literature
A: My lucky coin.
   Superstitious

Q: What do you miss the most about the
before times?
   nostalgic
A: All the people. It's lonely in the
wastelands.
   sentimental, gregarious
A: There used to be a lot more to do
around here. Everything is so boring
these days.
   playful, bored
A: My family. They were all killed in
the incident.
   mother dead, father dead, likes
mother, likes father, nostalgic,
grief_stricken
A: I had a dog. Now, I have nothing.
   pet dead, grief_stricken

Q: What do you think that you're better
at than you really are?
   arrogant
A: I like to think I'm pretty funny. No
one else seems to agree.
   obnoxious
A: I'm a really good singer! Probably.
   tone_deaf
A: I can weather any storm. As long as
it's not a metaphor for a difficult
emotional experience.
   vulnerable
A: I'm a talented leader; just, most of
the time, people ignore my guidance.
   bossy, ignored

Q: What could you have done to save
your brother's life?
   dead brother, family_guilt
A: Taken the car keys away
   alcoholic brother
A: Taken him away from dad
   abusive father
A: Made him move in with me to get him
out of the neighborhood
   brother_gang_member

Q: What book do you read every year on
the anniversary of your father's death?
   dead father
A: The bible
   religious
A: "Ender's Game".  He loved it.
   loves father
A: King Lear.
   narcissistic father

Q: Why are you the black sheep of the
family?
   black_sheep
A: I married outside of our faith
   religious_family
A: I just have this temper
   bad_temper
A: I like the bottle too much
   alcoholic self
A: I'm a fucking loser
   hates self

Q: How often do you see your family?
   living family
A: Once a year
A: Once a week
   loves family
A: Never
   estranged_from_family
```

References

Adams, T. & Adams, Z., 2006. Slaves to Armok: God of Blood Chapter II: Dwarf Fortress.

Alder, A., 2012. *Monsterhearts*, Buried Without Ceremony. Available at: https://buriedwithoutceremony.com/.

Barmore, N. et al., 2005. *Dread*, The Impossible Dream. Available at: http://www.tiltingatwindmills.net/.

Clocksin, W.F. & Mellish, C.S., 2003. *Programming in Prolog: Using the ISO Standard* 5th ed., New York, NY: Springer.

Evans, R., 2009. AI Challenges in Sims 3. In *Artificial Intelligence and Interactive Digital Entertainment*. Stanford, CA: AAAI Press.

Evans, R. & Short, E., 2013. Versu.

Gebser, M. et al., 2012. *Answer Set Solving in Practice*,

Gelfond, M. & Lifschitz, V., 1988. The stable model semantics for logic programming. *5th International Conf. of Symp. on Logic Programming*, pp.1070–1080.

Horswill, I., 2018. CatSAT: A Practical, Embedded, SAT Language for Runtime PCG. In *AIIDE-18*. AAAI Press.

Ingold, J., 2015. Adventure in Text: Innovating in Interactive Fiction. In *Game Developer's Conference*. San Francisco, CA: UBM Techweb.

LeBlanc, M., 2005. Tools for Creating Dynamic Game Dynamics. In K. S. Tekinbas & E. Zimmerman, eds. *The Game Design Reader: A Rules of Play Anthology*. Cambridge, MA: MIT Press, pp. 438–459.

Maxis, 2009. The Sims 3.

Morningstar, J., 2009. *Fiasco*, Durham, NC: Bully Pulpit Games.

Nelson, G., 2006a. Inform 7.

Nelson, G., 2006b. Natural Language, Semantic Analysis, and Interactive Fiction.

Nelson, G., 2013. *Writing for Versu*, San Francisco, CA: Linden Lab.

Nelson, M. & Smith, A., 2016. ASP With Applications to Mazes and Levels. In N. Shaker, J. Togelius, & M. J. Nelson, eds. *Procedural Content Generation in Games*. Berlin, Heidelberg: Springer, pp. 143–158.

Smith, A., 2011. A Map Generation Speedrun with Answer Set Programming. *Expressive Intelligence Studio Blog*. Available at: http://eis-blog.ucsc.edu/2011/10/map-generation-speedrun/.

Smith, A.M., Andersen, E. & Mateas, M., 2012. A Case Study of Expressively Constrainable Level Design Automation Tools for a Puzzle Game. In *International Conference on the Foundations of Digital Games*. Raleigh: ACM Press.

Smith, A.M. & Mateas, M., 2011. Answer Set Programming for Procedural Content Generation : A Design Space Approach. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3), pp.187–200.

Stark, L., 2014. *Pocket Guide to American Freeform*, CreateSpace Independent Publishing Platform.

Inkle, 2013. Ink.

Unity Technologies, 2004. Unity 3D.