Germinate: A Mixed-Initiative Casual Creator for Rhetorical Games

Max Kreminski,1 Melanie Dickinson,1 Joseph C. Osborn,2
Adam Summerville,3 Michael Mateas,1 Noah Wardrip-Fruin1

1University of California, Santa Cruz, 2Pomona College, 3California State Polytechnic University, Pomona

Abstract

Digital games are hindered as an art form by significant technical barriers to entry, which exclude many would-be game developers from participating in this medium of expression. Casual creators for game design attempt to mitigate these barriers, but—like conventional game development tools—often require users to “work their way up” from low-level mechanics to high-level rhetorical or expressive goals, rather than allowing them to start with high-level rhetorical goals and “work their way down.” The constraint-based game generator Gemini is well-suited to the generation of games that meet high-level expressive goals, but is difficult for casual users to work with. We present Germinate, a mixed-initiative casual creator for rhetorical games that extends Gemini with a more approachable graphical user interface. A preliminary expert evaluation revealed that Germinate affords a playful approach to rhetorical game design by generating games that successfully meet user intent in surprising and novel ways.

Introduction

Making videogames requires significant technical knowledge, and the technical barriers to entry associated with game creation exclude many would-be game creators from participating in digital game design as a medium of expression. Both the gulf of execution (the difficulty of translating one’s creative intent into executable code) and the gulf of evaluation (the difficulty of evaluating a work-in-progress game design and determining how it could be improved) (Hutchins, Hollan, and Norman 1985) contribute to the overall difficulty of making games.

Furthermore, the goal of game design is usually to produce some sort of experiential, rhetorical, or aesthetic effect in the player, but it is not possible to modify a game’s aesthetic effects directly. Instead, game designers may only directly modify mechanics, which interact with one another to produce emergent dynamics, which in turn interact with players to produce aesthetics (Hunicke, LeBlanc, and Zubek 2004). This has led to the development of the theory of procedural rhetoric (Bogost 2007), which attempts to show how changes in mechanics lead to changes in a game’s perceived meaning.

Casual creators (Compton and Mateas 2015; Compton 2019) for game design—approachable digital game creation tools that emphasize the process of exploring a game design space over the production of fully realized games—attempt to mitigate the difficulty of making digital games. These tools typically target users who are experts in a particular domain but inexpert as game designers, such as journalist creators of newsgames (Treanor and Mateas 2009), or users who are interested in creating and sharing games primarily as a form of casual self-expression, such as game-literate but non-designer players of games. These users can benefit substantially from the creativity support that casual creators provide, and they may not know how to craft low-level game mechanics that facilitate the communication of their expressive or rhetorical intent through game design.

However, most casual creators for game design still require users to build games up from low-level specifications of mechanics and individual pieces of content, rather than allowing users to directly express their rhetorical intent to a system capable of generating games that fulfill this intent. If the game designer’s end goal is to achieve a rhetorical effect on the player, as is often the case with newsgames and other personal (Anthropy 2012) or persuasive (Bogost 2007) games, then a rhetoric-forward approach to design may be preferable over an approach that starts with mechanics—especially for initial exploration of how an argument might be expressed through procedural rhetoric, before the game designer has decided what mechanics they want to pursue.

In this paper, we present Germinate, a mixed-initiative co-creative (Liapis et al. 2016) casual creator for rhetorical game design. Germinate is open-source1 and can be used in a web browser. Germinate is built on Gemini (Summerville et al. 2018), an abstract game generator that operationalizes the theory of proceduralist readings (Treanor et al. 2011) for game generation, and provides additional scaffolding for the basic interaction loop of working with Gemini in order to make it more approachable to casual users. In the remainder of this paper, we first discuss related work in the areas of rhetoric-driven game generation and casual creators for game design. We then describe how Germinate works as a system. We discuss how Germinate makes use of casual cre-

---

1https://github.com/ExpressiveIntelligence/Germinate
ator design patterns, and present the results of a preliminary expert evaluation showing how Germinate functions as a casual creator. Finally, we discuss takeaways that might be applicable to the development of future mixed-initiative casual creators for game design.

Related Work

The Game-O-Matic system (Treanor et al. 2012) represents an early landmark example of both a casual creator for game design and a rhetoric-forward approach to game generation. Game-O-Matic is unusual among game generators in its presentation of an interactive interface for human-in-the-loop game generation, which allows users to specify their rhetorical intent for a game as a diagram containing both nouns (i.e., named entities that must appear in the generated game) and verbs (i.e., a fixed set of abstract mechanical relationships, such as “chases”, “makes”, and “helps”, that one entity may have with another). Verbs are represented by directed arrows from one noun to another; any given pair of nouns can be related to one another by zero or more verbs. Users may also choose the sprites by which each noun should be represented in the generated games. When the user is satisfied with their intent, they instruct the generator to produce a game for them. They can then play the generated game directly within the Game-O-Matic user interface to determine what they like about it, what they dislike, and how they want to modify their intent (if at all) in response to the generated game.

The abstract game generator Gemini is based on an operationalization of the theory of proceduralist readings, and—like Game-O-Matic—takes a rhetoric-forward approach to generation. Gemini simultaneously generates game rulesets and proceduralist interpretations of those rulesets via answer set programming (ASP), a constraint satisfaction framework based on first-order logic. Gemini users provide the generator with a design intent, a short AnsProlog program that provides additional constraints on the overall generative space of Gemini games. The constraint solver proves that all of the interpretations demanded by the design intent are satisfied in every generated game. Therefore, only games with internally consistent proceduralist readings can be generated. Gemini generates games in the Cygnus game description language—a strict superset of the popular Video Game Description Language (VGDL) (Ebner et al. 2013; Schaul 2013)—and has a more sophisticated model of games than Game-O-Matic, so it can generate games that Game-O-Matic can’t. For instance, while Game-O-Matic only supports relationships between entities, Gemini also supports named resources; relationships between entities and resources; and precise low-level specifications of mechanics. Gemini also supports much more complex and expressive constraints on the generative space than Game-O-Matic allows, because Gemini design intents can make full use of AnsProlog language constructs to further constrain the generative space in arbitrary ways.

Many past approaches to game generation (Liapis et al. 2018; Khalifa et al. 2017) make use of evolutionary algorithms, attempting to generate progressively better games through incremental optimization of a utility function. This approach has been broadly successful for open-ended game generation, but in the context of rhetorical game generation, it is difficult to specify a utility function representing the extent to which a game instantiates a particular argument through its procedural rhetoric. In contrast, Gemini captures the specification of desired rhetorical qualities in a game by inferring whether those qualities follow from a set of mechanics using ASP. The inferential chains can be quite complex, and are not easily captured by evaluation functions that measure properties of rollouts of the game given an artificial player, or through static, numeric evaluation of the mechanics present in a game. The ANGELINA 3 and 4 systems (Cook, Colton, and Gow 2016a; 2016b) take an evolutionary approach, but also attempt to theme their games in meaningful or rhetorical ways through the judicious selection of appropriate assets. These themes are, however, limited in the extent to which they engage with the procedural rhetoric of the game’s rules and mechanics to create meaning.

Other casual creators for game design that don’t make use of game generation directly include Wevva (Powley et al. 2017) and the other work under the label of “fluidic games” (Nelson et al. 2017). These systems work by defining a large game design space characterized by hundreds of numerical parameters and presenting users with an approachable interface for exploring this space. This approach has been successful in enabling game creation across broad audiences. From a rhetorical game design perspective, however, these systems—like many other game creation tools—leave it up to the user to craft or discover low-level mechanics that create the desired meaning.

System Description

Germinate is a mixed-initiative casual creator for rhetorical games, similar to those produced by Game-O-Matic (Treanor et al. 2012). Germinate extends the underlying Gemini game generator with a browser-based graphical user interface that allows users to specify the properties they would like the generated games to have. Once the user has specified their desired game properties, they request a batch of games. Germinate translates these properties into a Gemini intent (an AnsProlog program that specifies additional constraints on the overall Gemini possibility space) and passes this intent to the underlying generator, which begins incrementally generating a batch of games. As games are generated, they are sent back to the Germinate frontend (figure 1), which allows the user to navigate through the pool of generated games; play the games in their web browser; and view the rules of these generated games, displayed as cards similar to those the user may use to specify their intent.

The overall interaction loop of working with Germinate is as follows:

1. Specify some initial constraints on the generative space.
2. Request a batch of games.
3. If the intent contains contradictions, go back and change it to eliminate the contradictions, then repeat Step 2; else go on to Step 4.
4. Inspect the generated games, by playing them and reading the rules.
5. Modify the constraints on the generative space to reflect your improved understanding of your own intent.
6. Repeat from Step 2.

This mirrors the interaction loop we found to be most effective when working with Gemini in general, but provides additional scaffolding around intent editing and evaluation of generated games. On the intent editing side, we mitigate the gulf of execution by providing users with a more structured interface for modifying the intent. This makes it harder for the user to accidentally produce a malformed intent, whether due to AnsProlog syntax errors or incorrect specification of Cygnus predicates—both of which are common when editing Gemini intent files directly. On the evaluation side, we mitigate the gulf of evaluation by providing users with live, playable versions of generated games directly in line with the intent that produced them. A similar interaction loop is also common when authoring answer set programs in general (Brain, Cliffe, and De Vos 2009).

The Germinate Intent Language

Germinate breaks down design intents into four types of cards: entity cards, resource cards, relationship cards, and trigger cards.

Entity cards describe Gemini entities: graphical objects with names and sprites that move around within the two-dimensional physical space of the game’s playing field, potentially responding to user input and interactions (such as collisions) with other entities.

Resource cards describe Gemini resources: named quantitative values that are displayed to the player in resource bars, and in terms of which all the game’s goals are ultimately framed.

Both entity and resource cards in Germinate may be annotated with a variety of appropriate tags, which constrain certain properties of the entities and resources to which they are attached. For instance, an entity may be given the “Quantity several” tag to specify that it must be possible for multiple instances of this entity to exist at any given time, or the “Controlled by player” tag to specify that this entity must respond directly to player controls; a resource may be given the “Initially low” tag to specify that its value must start out nearly empty; and both entities and resources can be given the “Player thinks it’s good” tag to specify that Gemini must read this entity or resource as good for the player overall.

Relationship cards do not map directly to a single underlying concept in Gemini. All relationships consist of a relationship type, a left-hand side, and a right-hand side. Depending on the relationship’s type, the left-hand and right-hand sides of the relationship may be entities, resources, or both. They may also be left blank, which signals to the generator that this side of the relationship should be filled in during generation. (For instance, a Ghost produces relationship would indicate to the generator that the Ghost entity must be read as producing some sort of resource, but which resource this actually is is left up to the generator.) Most relationship types correspond to Gemini readings (such as the produces, consumes, and helps readings), but some (such as the collides with relationship type) correspond to outcomes instead.

Trigger cards map closely to Gemini “outcomes”, which tie together one or more preconditions (situations that might emerge during gameplay) with one or more results (effects that happen when these preconditions are met).

The space of Gemini design intents that Germinate is capable of producing is a strict subset of the full Gemini intent space. As a result, Gemini is capable of producing many games that Germinate cannot. However, the restrictions that Germinate places on the intent space support casual use in several different ways. First, these restrictions focus users on exploring the most well-tested subset of Gemini’s readings, reducing unpleasant surprises arising from poorly-tested interactions of Gemini game features. Second, restrictions mitigate fear of the blank canvas (Kremsinski and Wardrip-Fruin 2019) by ensuring that even an empty or trivial initial intent can be used to produce mostly-random games that provide concrete inspiration for further intent refinement. And third, restrictions mitigate the temptation to precisely specify all game rules and interactions by hand as part of the intent, promoting the beneficial forms of surprise that we identified during preliminary expert evaluation of Germinate.

Germinate/Gemini Translation

The differences between the Germinate and Gemini intent languages require translation to be performed at two key places in the interaction loop. When the user submits their design intent to the generator for consideration, it must first be translated from a Germinate intent (a set of cards specifying desirable game properties) to a Gemini intent (an answer set program constraining the Gemini generative space in ap-
The generator can understand, Germinate must also trans-
click_on_student_for_knowledge :-
    initialize(set_value(
        entity(student).
    :- not reading(book,good).
    entity(book).
    :- not reading(student,good).
    resource(knowledge).
    :- not reading(knowledge,good).
    resource(tiredness).
    :- not reading(tiredness,bad).

Some intents translate straightforwardly from the Ger-
mate to the Gemini intent language. For example, one eval-
ator of the tool created an intent stating that the generated

games should contain two entities (books and students); two
resources (knowledge and tiredness); and that books, stu-
dents, and knowledge should be perceived by players as
good, while tiredness should be perceived as bad. Each of
these concepts can be expressed directly as a single Gemini
intent predicate, so the intent translates directly into the fol-
lowing AnsProlog code (simplified slightly for readability):

entity(book).
    :- not reading(book,good).
entity(student).
    :- not reading(student,good).
resource(knowledge).
    :- not reading(knowledge,good).
resource(tiredness).
    :- not reading(tiredness,bad).

Some intents, however, are slightly more difficult to trans-
late. Intents involving negation often require more work
to transform into AnsProlog code, as do intents involving
higher-order concepts that Germinate exposes to users di-
rectly but that map to a more complicated set of Gemini in-
tent predicates under the surface. Suppose a user wants to
add some new restrictions to the previous intent: books must
be perceived by the player as either good, bad, or neutral, but
not “complicated” (i.e. both good and bad); students must
have the “collides with” relation to one another; the tired-
ness resource must not start off empty; and the mechanics
must not involve clicking on students to increase knowledge.
This adds several more complicated integrity constraints to
the AnsProlog program:

    :- not 0 { reading(book,good;bad) } 1.
    :- not result(_,apply_restitution(
        student,student)).
    :- initialize(set_value(
        tiredness,scalar(0))).
    click_on_student_for_knowledge :-
        precondition(
            control_event(click(student)),0,
            result(0,modify(increase,knowledge))).
    :- click_on_student_for_knowledge.

In addition to translating design intents into a form that
the generator can understand, Germinate must also trans-
late generated games back into the Germinate interface lan-
guage. Games generated by Gemini (i.e. game rulesets ex-
pressible in the Cygnus game description language) con-
tain a range of assertions, including some that Germinate
does not yet know how to interpret or present to the user.
To give one example, the first game generated by Gemini
for the simple evaluator-created intent described earlier in
this section consists of 76 total statements, many of which
pertain to inferences that Gemini has made about how this
game’s procedural rhetoric can be “read” by the player. Ger-
mate recognizes and displays to the user in some form 39
of these statements: 8 pertaining to entity and resource tags,
including the reading of entities and resources as “good” or
“bad”, the quantities of entities, and the initial values of re-
sources; 6 pertaining to readings of relationships between
entities and resources, including “produces”, “consumes”,
“helps”, and “shares” relationships; and 25 pertaining to pre-
conditions and postconditions on generated game mechan-
ics, or “triggers”. All of these Germinate-recognized facts
about the generated game can then be imported back into
the intent in some form, if the user decides to further explore
this part of the design space.

Evaluation

To evaluate Germinate, we employed two kinds of eval-
uation. First, we performed a principles-based evaluation,
treating the casual creator design patterns identified by
(Compton and Mateas 2015) as a set of design principles.
Second, we performed a preliminary expert evaluation with
four researchers in the field, all of whom have experience
working with game generators other than Gemini.

Casual Creator Design Patterns

(Compton and Mateas 2015) define eleven design patterns
for casual creators, five of which are present in Germinate.

No blank canvas. Germinate is capable of generating

games regardless of whether the user has specified any prop-
erties in the intent, meaning that users can always rely on the
system to provide inspiration. Additionally, Germinate often
generates games that contain additional, inspiring random
elements beyond those specified by the users.

Limiting actions to encourage exploration. Users of

Germinate are restricted to a relatively narrow subset of the
full Gemini intent language, which was chosen to steer users
toward a known-good subset of the overall generative space.

Mutant shopping. Germinate produces a few distinct
variations on each base game that Gemini creates in re-

sponse to the user-supplied intent, allowing users to scan
through these variants for inspiration and import the desir-
able features of multiple generated games into a single com-
bined intent.

Modifying the meaningful. Germinate foregrounds
specification of high-level rhetorical intent over granular
specification of low-level mechanics, although both modes
of interaction are available.

Simulation and approximating feedback. To assist

users in understanding the generated games, Germinate
parses mechanics and Gemini-emitted “readings” from gen-
erated games and displays them in card format. This allows
the user to read these cards to understand what arguments the generator is “trying to make” and how the rules are structured, providing additional approximating feedback beyond what a user could learn from playing the generated games.

Of the remaining six casual creator design patterns, we found that two (Instant feedback and the Chorus Line, the latter of which is a “sub-pattern” of the former) are technically infeasible to realize in the current version of Germinate, due to how long Gemini takes to produce games once given an intent. Two more (Entertaining evaluations and Saving and sharing) are technically feasible but not yet implemented in Germinate. And two others (Hosted communities and Modding, hacking, teaching) are community-oriented design patterns that are unsupported by the current version of Germinate, which focuses on “single-player” features alone.

Preliminary Expert Evaluation

Germinate is meant to be used by people with a variety of backgrounds. There are several ways we could evaluate the tool—for example, whether it makes Gemini experts more efficient at producing games achieving their design goals, whether Germinate is usable from a traditional task-oriented HCI standpoint, or whether non-experts find the tool pleasant and diverting to use. In order to rapidly gather feedback and validate our approach before taking on a larger user study, we chose to focus our preliminary evaluation on overall impressions of the tool and the sense of agency experienced by users familiar with game generation but not deeply familiar with Gemini while using the tool.

Emma’s Journey (Garbe et al. 2019) is an experimental narrative game that juxtaposes choice-based narrative scenes with abstract minigames generated by Gemini. The creators of Emma’s Journey used scene-specific Gemini design intents to generate games whose procedural rhetoric matches and contributes to the intended procedural rhetoric of each scene. As an initial assessment of Germinate’s capabilities, two co-authors familiar with Gemini attempted to reproduce games from specific Emma’s Journey scenes using Germinate, working from the same prompts originally used to hand-author Gemini design intents as obtained from the creators of Emma’s Journey. We found that while we were able to get close to the goal intent within a few minutes (comparable to the time experts might spend hand-authoring), we could not exactly phrase the intent goals due to the subset of Gemini exposed by Germinate. On the other hand, the resulting games were surprising and interesting, branching off in ways that were both unexpected and sometimes preferable. Moreover, Germinate’s constraints led us to explore a broad variety of ways to translate the given intents to Germinate’s intents language.

It is important to note that this is not the primary mode of use intended for Germinate, which as a casual creator tool is meant to be more exploratory. However, we do feel that this preliminary assessment is informative, suggesting that Germinate is competitive with hand-authored design intent files even in the hands of Gemini experts.

For our main evaluation, we enlisted three Ph.D. researchers in game generation (not deeply familiar with Gemini) and one post-bac researcher. After a 5 minute walkthrough of Germinate’s user interface and ontology, we instructed each subject to play around with the tool for 10 minutes and talk through their thinking and reasoning. If the user did not say anything for a period of time, we would prompt them. Afterwards, we administered the following brief user experience questionnaire (Laugwitz, Held, and Schrepp 2008), of which the first question was free response and the remaining questions were on a three-point scale:

1. What is your overall impression of the tool?
2. How easy was it to use the tool?
3. Were you able to use the tool without unnecessary effort?
4. Did you feel you had enough control over the generated games?
5. Was it fun to use?

Results

Our first respondent (the post-bac researcher) felt that overall, Germinate was intended for game creators who wanted to drive game design by story or environment elements rather than specific rules or mechanics. They especially liked to play with the randomization features. They described the tool as “easy to use” (2) (although waiting for games to be generated was frustrating), stated that “some unnecessary effort” (3) was involved (especially when the generated intent was not satisfiable for one reason or another), and they felt “some control” (4) over the generated games. This user especially appreciated that the system pushed them away from traditional game rules and mechanics and towards more surprising areas of the generative space. Finally, they felt that the tool was “fun to use” (5), commenting specifically on the evocative use of emoji and the loose, not overly goal-oriented interaction style. They also believed that given more time with the tool, they could learn how to exert more influence over, for example, the control schemes used for the game entities.

A researcher in playable AI systems found Germinate “an interesting casual creator for game generation,” and engaged with the tool “without any specific goals.” They mainly worked by defining entity and resource tags and relationships, but felt uncertain about the concrete meaning of terms used in the interface. Their main frustrations concerned the unclear rules of generated games; still, they drew out semantic arguments unprompted (“my Friend is chasing Success”). They found the tool neither easy nor hard to use (2), since the tool can’t be meaningfully separated from the games and affordances of the generator. They did not experience a sensation of unnecessary effort (3), and since they had no specific goals they felt “full control—as much control as [they] were looking for;” but perhaps not enough if they intended to create a specific game (4). Finally, they felt the tool was fun to use (5) and experiment with as a casual user (as opposed, say, to a creator of newsgames).

A game generation researcher felt that the tool is a “good step,” but felt that the games were too similar to each other. They found the tool “hard to use” (2) most likely because they felt that they had “some/no control” over the generated
restricted than that of Gemini, we nevertheless preserve the
eneration experience as well. That the same holds for casual users with no prior game gen-
envisioned game. Further evaluation will be needed to show
ward precise specification and realization of a single, clearly
ration of a sometimes-surprising generative space than to-
rhetorical game design, oriented more toward playful explo-
users in its overall goal of facilitating a casual approach to

Figure 2: A generated game that surprised one of the ex-
pert participants. The intent states that books, students, and
knowledge are good, while tiredness is bad. The participant
expected a game in which a student chased books, but the
resultant game featured a student warily circling the books
while the player tried to corral the books towards the student.

games (4). However, they did feel that the there was “no un-
necessary effort” in using the tool (3) and ultimately found
that the tool as “somewhat fun to use” (5) (although they
noted that they wished they had more time to play around
with the relationships and potential expressible space of the
system).

On the other hand, one of the game generation researchers
felt that generally, “[t]here’s a pleasant flow to using the tool,
and it was able to generate some games I wasn’t expecting
but really liked.” The intent and resultant game can be seen
in figure 2. They especially liked how Germinate generated
games that fulfilled the rhetorical intent without using the
mechanics they were expecting. They described the tool as
“easy to use” (2) (although they noted that there was a fric-
tion in how the rules are presented and how they wanted to
be able to tell the generator to change a specific rule), said
that there was “no unnecessary effort” (3) in using the tool,
felt that they had “full control” over the generated games (4),
and felt that the tool was “fun” (5) to use.

Discussion

All expert users surveyed in our evaluation found Germinate
to be at least “somewhat fun to use”, and three of four said
it was “fun” unreservedly. Three found that there was “some unnecessary effort.” Two of the expert
evaluators felt that they had “full control” over the generated
games, while the other two felt that they had “some control”.
Two found it “easy to use”, one “hard to use”, and one some-
where in between. Altogether, we conclude that—despite
some rough edges—Germinate succeeds among these expert
users in its overall goal of facilitating a casual approach to
rhetorical game design, oriented more toward playful explo-
ration of a sometimes-surprising generative space than to-
ward precise specification and realization of a single, clearly
envisioned game. Further evaluation will be needed to show
that the same holds for casual users with no prior game gen-
eration experience as well.

Although Germinate’s intent space is substantially more
restricted than that of Gemini, we nevertheless preserve the
desirable Gemini property of being able to generate games
that the earlier interpretation-driven game generator Game-
O-Matic could not. In particular, the Gemini intent language
allows users to specify named resources with their own prop-
erties, as distinct from entities; relationships between enti-
ties and resources; fine-grained specifications of an individ-
ual entity’s gameplay properties; and fine-grained specifi-
cations of diverse mechanics—none of which are supported by
Game-O-Matic, and all of which are made available to users
of Germinate.

Moreover, the finer-grained control that we sacrifice in
giving up direct editing of AnsProlog design intents is re-
warded here by greater approachability for casual users,
who could feasibly use Germinate to perform interpretation-
driven game generation without any knowledge of AnsPro-
log. This trading off of control for power is a common fea-
ture of casual creators, and can arguably be considered ben-
eficial insofar as it facilitates user reflection on intent.

The most common user frustrations we encountered in
our expert evaluation involved opaque interactions between
user-specified constraints. These interactions sometimes
resulted in the generation of unsatisfiable intents without any
feedback given to the users as to why the intent was unsat-
sifiable. User-friendly explication of how constraints inter-
act and why an unsatisfiable intent cannot be fulfilled—in
other words, improved error reporting—is therefore a top
priority for future work. It would be fairly straightforward to
modify Germinate to report some warnings to the user when
they submit an intent that is likely to be unsatisfiable due
to mutually incompatible constraints, especially since we’re
already logging some of these warnings to the console—
but this wouldn’t be sufficient to capture the full range of
mutually incompatible constraints you can specify with any
nontrivial fragment of the intent language. Moreover, it is in-
herently difficult to get the underlying Clingo solver (around
which Gemini is built) to report the reason an intent is unsat-
sifiable: there are often several distinct integrity constraints
that are all violated by a bad intent, making it difficult to
isolate the precise reason for the failure (Syrjänen 2006).
During debugging, we made use of a script that performs
a binary search over the core Gemini AnsProlog integrity
constraints (low-level constraints that capture Gemini’s def-
inition of what a game is) and repeatedly re-runs the solver
with some constraints relaxed to identify the constraints that
are most likely invalidating the current intent; this approach
could possibly be used to improve error reporting going for-
ward, but to do so would be nontrivial from both an engi-
neering and a user experience perspective.

Another possibility for future work involves exposing the
derivation trees of Gemini arguments to users directly and
allowing users to phrase their intents partly in terms of these
derivations. Gemini internally contains hundreds of individ-
ual rules that specify particular arguments as to why a game
with certain characteristics can be interpreted as possessing
a certain rhetorical property, and it seems (based on the ex-
pert evaluation we conducted here) that users often want to
be able to both understand why the generator is interpret-
ing this game in a particular way and inform the generator
that some of its internal interpretive strategies should not be
considered here. Exposing these derivation trees to the user, and letting the user specify in their design intent that some derivation steps should not be used, would perhaps restore some of the finer-grained control that users found the current version of Germinate to be lacking, without overwhelming users with the full complexity of the AnsProlog language.

**Conclusion**

We present Germinate, a casual creator for rhetorical games based on the Gemini game generator. Principles-based and preliminary expert evaluation shows that Germinate successfully affords an approachable and playful form of rhetorical game design, satisfying the user’s expressive intent in surprising but inspiring ways, and scaffolding both the execution and evaluation sides of the interaction loop. Larger-scale evaluation in the future will be necessary to show that these desirable properties hold for casual users as well.

**Acknowledgements**

This research was supported by a Seed Fund Award CITRIS-2017-0187 from CITRIS and the Banatao Institute at the University of California.

**References**


Cook, M.; Colton, S.; and Gow, J. 2016b. The ANGELINA videogame design system—part ii. *IEEE Transactions on Computational Intelligence and AI in Games* 9(3):254–266.


