Proceduralist Readings, Procedurally

Anonymous for Review

Abstract

While generative approaches to game design offer great promise, systems can only reliably generate what they can "understand," often limited to what can be hand-encoded by system authors. Proceduralist readings, a way of deriving meaning for games based on their underlying processes and interactions in conjunction with aesthetic and cultural cues, offer a novel, systematic approach to game understanding. We formalize proceduralist argumentation as a logic program that performs static reasoning over game specifications to derive higher-level meanings (e.g., deriving dynamics from mechanics), opening the door to broader and more culturally-situated game generation.

Introduction

Games make use of a broad range of communication mechanisms. Through sensory affordances, processes, and interaction rules, as well as standard semiotic systems from other media, including visual imagery, verbal language, and their interaction with human psychology, a game may communicate rich arguments and emotional experiences. For example, Molleindustria's *The Free Culture Game*[1] and Vi Hart and Nicky Case's *Parable of the Polygons*[2] each make political arguments by operationalizing human-driven processes like commodification and segregation as emergent dynamics arising from player interaction.

To understand the meaning of a game requires accounting for each of these mechanisms and where they interact. Treanor et al. proposed *proceduralist readings*, a process of deduction that allows one to discuss dynamics, aesthetics, and higher-level meanings on the basis of a game's mechanics and cultural knowledge about the significance of sensory cues like colors and icons (Treanor et al. 2011). They refer to the result of this process as a *meaning derivation*, a tree-structured logical argument for any given interpretation of what a game communicates.

The Mechanics, Dynamics, and Aesthetics (MDA) framework of Hunicke et al. (Hunicke, Leblanc, and Zubek 2004a) posits that at the base level games are composed of small *mechanics* that interact with each other to form the *dynamics* of the game. The *dynamics* of the game in turn affect a response in the player, i.e. the *aesthetics* of the game. In this work we present an operationalization of proceduralist readings that enables *automated game understanding*: a computational encoding of how to derive higher-level knowledge (dynamics) from lower-level knowledge (mechanics). Moreover, by incorporating cultural and phenomenological knowledge we can derive aesthetics from the inferred dynamics. This process is performed using Answer Set Programming (ASP) to statically derive all of these properties without need for simulation.

While this work focuses on the automated reasoning about games, our ultimate goal is to generate games from meaning specifications. By using a declarative modeling technique, ASP, we can create a body of reasoning principles whose operational behavior is *bidirectional*: the rules relate high-level meanings to game instantiations (detailed mechanics) without imposing a pipeline ordering from one to the other. We can hold the game instantiation fixed and use the rules to infer high-level meanings (a la static analysis over game mechanics)—the main result described in this paper—or, going forward, we can hold the high-level meanings fixed (design intents) to generate a game. We further discuss this ongoing work in the last section.

We contribute a formalized theory of deriving game dynamics from mechanics, as well as aesthetics from dynamics given cultural knowledge models. Among the dynamics derived are approximated player models based on static analysis of game rules.

Related Work

The most closely-related work is *Game-o-Matic* (Treanor et al. 2012a), which similarly combines cultural knowledge with mechanics to generate playable games from meaning descriptions. It makes use of "micro-rhetorics" to represent primitive mechanics paired with player interpretations of game entity verbs (e.g., *support*, *eat*, *destroy*, etc.). The work presented here generalizes from *Game-o-Matic* in a number of ways. First, *Game-o-matic* only represents and reasons about game rules related to movement and collision detection (*graphical logics* in operational logic parlance), while the work presented in this paper captures a broader range

[1] `http://www.molleindustria.org/en/freeculturegame/`

[2] `http://ncase.me/polygons/`

of operational logics. Second, its notion of game meaning is restricted primarily to single-layered mappings between mechanics and represented game verbs, with no explicit reasoning related to game dynamics. In contrast, the work represented in this paper operationalizes meaning derivations to capture both broader and more abstract notions of dynamics and game meaning. Finally, *Game-o-Matic* is a pipelined architecture that can move from a game specification in the form of game entities with desired verb relationships to a game, while the work presented in this paper uses a constraint satisfaction framework to be able to move from desired dynamics and meanings to games, or games to inferred dynamics and meanings, in a single framework. This paper has focused on the latter, which can be considered a generalized form of "reverse *Game-o-Matic*."

Prior work in game generation also deals with inferring relationships between formalized games and meanings or constraints (the inputs to the generator). One game generation system (Nelson and Mateas 2007) incorporates meaning through WordNet and ConceptNet and characterizes games in terms of patterns found in their generation target, WarioWare games, such as *avoid* and *acquire*. Our system can express a broader, finer-grained space of mechanics, and express meaning in more systematic terms. The *Variations Forever* (Smith and Mateas 2010) system similarly uses ASP to search a fine-grained space of possible mechanics but does not make any attempt to relate generation to high-level conceptual meaning, nor is the generative space broad enough to express general resource or non-graphical-logic games. Finally, the *Angelina* system has generated games based on theme in the context of the Ludum Dare game jam (Cook and Colton 2014). Angelina uses a data-driven approach based on word association databases and does not systematically relate mechanics to meaning.

Reasoning over the structures of a game has also been a topic of research for General Game Playing (GGP) systems. The earliest work (Kuhlmann, Dresner, and Stone 2006) examines the syntactic structures of the first-order logic program that defines a GGP game to learn underlying game structures. This system could answer very basic questions, e.g., "Is this a successor function?" or "Is there a board?" Later work by Schiffel and Thielscher (Schiffel and Thielscher 2007) extended this work to be able to determine if something represents a quantity (such as number of points). Due to the fact that the players are given no type information and very limited semantic information for the logic programs that make up a GGP game, even these basic tasks are arduous—determining if there are numbers or a board has to be teased out from the syntax. Our formalism lends itself to discovering higher level structures due to the fully-specified semantics.

Approach

The originators of the term "proceduralist reading" demonstrate, through example, the process of constructing meaning derivations, or bodies of knowledge about a game that are built up in a proof-tree-like structure from axiomatic facts such as mechanics and thematic elements (Treanor et al. 2011). The lowest-level inferences of a meaning derivation, e.g., A *collides with* B, synthesize time-varying audiovisual content (moving images which change behavior when overlapping) with subjective expectations of simulated systems (2D physical space, agents). Further inference of A *eats* B may be possible by combining cultural knowledge and simulated behavior, for example if A appears as a predatory animal and B as prey and B shrinks or disappears when touched. The former type of inference connecting observable phenomena to the evolution of the system's implied simulation is directly supported by one or more *operational logics* (Wardrip-Fruin 2005; Mateas and Wardrip-Fruin 2009), in this case a *collision logic*. The latter type leverages an *interpretive affordance* (eating) satisfied by both the theming and simulation behavior.

Given the language of operational logics, one may define mechanics (e.g., "the cursor exerts a force on particles"), atomic pieces of cultural knowledge (e.g., "the color green represents life"), and communication strategies (e.g., "a meter display communicates the state of a resource"). From these specified knowledge components, the task of game understanding is to derive *higher-order knowledge* like "the ball moves perpetually back and forth unless the player intervenes," an example of deriving a dynamic from mechanics in the language of the MDA framework (Hunicke, LeBlanc, and Zubek 2004b). Even the isolated task of deriving dynamics from mechanics has seldom been addressed in previous computational systems.

Proceduralist readings provide a solid foundation for the human activity of game understanding, in which a reader thinks critically about which pieces of the game offer which interpretive affordances. We build on this work by formalizing this reasoning as a computer program such that we can autonomously derive high-level knowledge, including culturally-informed critical readings, from an input game description.

Explanatory example

One example carefully studied by Treanor et al. is *The Free Culture Game*, in which "new ideas" are represented as floating particles that must be herded towards producers in the creative commons to keep them inspired (creating new ideas) and away from the *vectorialist*, who takes ideas out of the creative commons to commodify them for consumers. The player exerts an indirect force via the mouse cursor on new ideas. Several proceduralist readings are extrapolated from these mechanics together with the game's interpretive affordances, such as the colors selected for producers versus consumers (green versus grey) and the robotic and malicious audio-visual character of the vectorialist. For example, they read the following meanings from the game:

1. The player must navigate the cursor between the vectorialist and new ideas to prevent commodification.
2. The vectorialist is an evil adversary who does not care about the happiness of people.

They derive these meanings from a number of implicit rules, which they call *dynamics*. We read these as equivalent to (Salen and Zimmerman 2004)'s *constitutive mechanics*,

though we will use the term *dynamics* here to avoid confusion. For example, the first inference shown on the path to deriving Meaning 1 is:

Because producers need new ideas to collide with them in order not to turn into consumers, the player's goal is to maintain as many producers as possible, and the player can exert a force on the new ideas, the player will push new ideas towards producers. (Dynamic 1)

Figure 1 depicts the tree structure of this argument with the conclusion at the root, premises as subderivations, and definition components (mechanics, in this case) at the leaves. While these reasoning structures constitute an initial step towards formalism, they need further development to constitute a general, computable theory. Even this simple derivation of a dynamic from mechanics requires quite a few more intervening steps to close all of the gaps with formal inference. Specifically, formal inference systems require that each step of reasoning (i.e., each horizontal line in the diagram) be justified by some general principle that we are applying to the specific terms of our argument. Such principles might be characterized through rules like:

- If the player's goal is G and action A accomplishes G, the player will do action A.
- Pushing an entity E with an entity P causes E to move away from P.
- An entity moving away from something might move toward another entity.
- When E moves toward X, E and X might collide.

Each of the variables in the rules described above is implicitly universally quantified, such that the formalization of each rule is simply a first-order logical formula, e.g., $\forall G, A.\ goal(G) \ \wedge\ accomplishes(A, G) \ \Rightarrow\ playerWillDo(A)$

Our formalization of proceduralist arguments thus consists of two pieces: a specification of the game's mechanics and communication choices (hereafter "the specification") and a collection of general rules like the above for reasoning over specifications (hereafter "the reasoning principles"). We encode both of these pieces in an Answer Set Programming (ASP) system, i.e., as a logic program interpretable by an answer set solver.

Operational Logics

The logical statements above refer to several independent domains: a player model that assumes the player wants to achieve the game's goals; a physics simulation described in terms of forces exerted; entity movement and collision tied to the graphical rendering of sprites on the screen; and the increase and decrease of quantities like a producer's current "inspiration" level. We can describe these domains in terms of operational logics (OLs); since proceduralist readings are grounded in OLs, assigning them a logical semantics here is a natural move. Operational logics provide a common vocabulary to address diverse types of interactive media, games included. As formulated in (Mateas and Wardrip-Fruin 2009), each OL provides both a set of *abstract operations* and a *communicative strategy* for showing interactors the game state over which the operations work and the activity of those operations proper.

In the Free Culture game, we see, among others: a *resource logic* with abstract operations like increasing or decreasing the happiness of each producer, communicated by color tones; a *collision logic* governing the collisions of entities, communicated by drawing them on a 2D canvas; a *character-state logic* whose operations include switching the movement behavior of the vectorialist or creating and destroying producers; a simple *movement physics logic* defining and enacting equations of motion for each entity; and a *control logic* which connects the mouse input to the movement of the cursor. Taken together, this combination is traditionally called a "graphical logic," though strictly speaking that term is a shorthand for any operational logic which communicates primarily through graphical channels (as opposed to e.g., textual ones).

We further subdivide the abstract operations of an OL into *observations* (like determining whether two objects overlap, or whether a resource quantity exceeds a threshold) and *changes* (like dynamically altering which objects an entity collides with, or causing a resource to increase in quantity). Our ASP encoding reifies these abstract operations and their connections to visible game state, yielding the atoms of which game mechanics are composed.

With OLs as our primitive constructs, we can describe game mechanics as a collection of named *outcomes* that have preconditions and results, similar to linear logic-based game specification in Ceptre (Martens 2015) or the sensors and actors of Kodu (Stolee and Fristoe 2011). Preconditions and results map onto the observations and changes, respectively, of OL transactions. In the next section, we explain this formalization language by example.

Examples

Our formalization efforts currently consist of a 900-line body of rules which are able to derive higher-level meanings from game specifications. Here we will discuss two examples to illustrate the breadth of the approach: one drawing from the existing body of "graphical logic" games, specifically *The Free Culture Game*; and the second drawing from a game without movements and collisions, specifically a miniaturized version of the browser-based incremental game *Cookie Clicker*.

The Free Culture Game

First, we describe our encoding of the Free Culture Game and then describe which rules in our reasoning system are needed to derive the example from the previous section.

```
%% Mechanic (4): The vectorialist pulls in new ideas.
precondition(near(vectorialist, new_idea), pull_idea).
result(pull_idea, move_toward(new_idea, vectorialist)).
```

This pair of rules establishes that `pull_idea` is an action that may occur in the game, that its sole precondition is that the vectorialist is near a new idea, and that its result is for the new idea to move toward the vectorialist. The predicate `near(vectoralist, new idea)` is an example of a condition and the predicate

goal is to maintain producers

producers convert when `ideasAbsorbed` < threshold

goal is to maintain `ideasAbsorbed`

`collide`(producer, new_idea) increases `ideasAbsorbed`

player wants `collide`(producer, new_idea)

cursor pushes new ideas

Dynamic 1: player will push new ideas toward producers

Figure 1: Meaning derivation tree for Dynamic (1) in The Free Culture Game. Base logical assertions are used in conjunction to derive the assertion below the line (i.e. from "goal is to maintain producers" and "producers convert when `ideasAbsorbed` < threshold" we infer "goal is to maintain `ideasAbsorbed`".

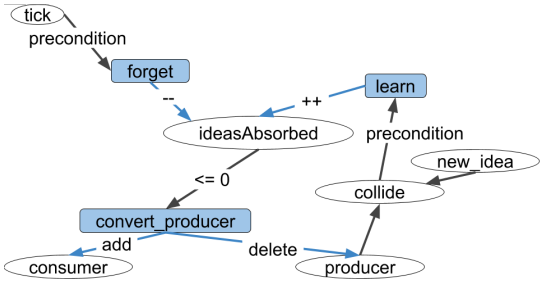

Figure 2: Mechanics interactions in *The Free Culture Game*. We represent outcome names as filled nodes and causal influence as directed edges.

`move_toward(new_idea, vectorialist)` is an example of a result.

The following three mechanics establish the relationships between producers, new ideas, and the player:

```
%% Mechanic (1): Producers make new ideas
precondition(slow_timeout, gen_idea).
result(gen_idea, add(new_idea)).

%% Mechanic (2): The cursor exerts force on ideas.
precondition(near(cursor, new_idea), push_idea).
result(push_idea, move_away(new_idea, cursor)).

%% Mechanic (6): Collision between new idea & producer
%%   increases ideasAbsorbed.
precondition(collide(new_idea, producer), inspire).
result(inspire, increase(ideasAbsorbed, mid)).
```

These mechanics and the relationships they induce between entities and resources may be better understood through visualization as in Figure 2.

Reasoning Principles Next, we need to introduce rules for deriving consequences of mechanics knowledge. We generate logical knowledge of game dynamics on the basis of mechanics plus *generic knowledge* about how mechanics relate to one another through the progression of time. Again, we interpret game dynamics precisely as *constitutive mechanics*, those "logically implied by the game's mechanics" (Nelson and Mateas 2009). Of course, "what follows logically" from a game's dynamics depends on the logical theory in place. While the principles of first-order logic allow us to instantiate generalizations with specifics and apply implications to known premises, exactly the content of those generalizations and implications remains a contingent body of knowledge to be authored. It requires logical modeling of game phenomena and their relationship in terms of the causal phenomena we wish to model.

For example, to derive a dynamic that "the player will do" something, we need to know that the action (a) will create a favorable condition and (b) is controlled by the player. We write such a rule in logic program notation as:

```
playerWillDo(Cond, Outcome) :- playerCreatesCondition(Cond),
                               conditionEnables(Cond, Outcome),
                               outcomeFavorable(Outcome).
```

We also give meaning to the operators of operational logics in terms of how they link together, i.e., how the results of operators in each OL enable certain conditions (in the same or other OLs). For example, two entities moving toward each other may cause the "collide" condition between them, and increasing a resource may eventually cause it to satisfy the condition of being over a certain threshold:[3]

```
result_enables_condition(move_toward(E1, E2), collide(E1, E2)).
result_enables_condition(move_away(E1, E2), collide(E1, E3)).

result_enables_condition(increase(Resource), Condition)
    :- high_threshold(Condition, Resource).
result_enables_condition(decrease(Resource), Condition)
    :- low_threshold(Condition, Resource).
```

In this regard, the game's specific rules and the general reasoning principles form two halves of a game's meaning: the game specification connects conditions to results (through explicit mechanics) and the reasoning principles connect results to conditions (through a semantic, approximated interpretation of each result).

To connect high-level ideas like *an outcome is favorable* with low-level models of game world domain knowledge, we introduce a number of rules resembling *static analyses* over a game's specification code. For example, we look at which conditions and results enable each other and take their transitive closure. Positive and negative valence—whether, e.g., an outcome or resource is considered favorable—is derived from this static analysis information (along with some axiomatic cultural assumptions). These valences extend recursively to relate to entities and player goals, which then inform the player model and connotation of other game agents as harmful or helpful.

Thematic and Cultural Information In order to derive aesthetics and culturally-informed meanings, we represent some assumptions about mappings between audiovisual stimuli and conceptual notions (like mood and political connotation) that we wish to reason over. For example, some

[3] Some rule preconditions have been elided here for brevity.

relevant assumptions for *The Free Culture Game* are that the color green represents a more healthy, positive color for a producer; that producers and consumers are shaped like people, which has a positive connotation; that black can connote evil, and that the vectorialist image is represented by a machine-like, black image. We include assumptions of this form in a separate module of predicates, allowing us to run the analysis and observe the resulting interpretation with and without these assumptions.

Example Output The answer sets generated by ASP when given the game specification, our rules, and our cultural reasoning module produce the following as a notable subset of facts derived:

```
goal(maintain(ideasAbsorbed))
outcomeAffects(ideasAbsorbed,forge,negative)
outcomeAffects(ideasAbsorbed,learn,positive)
outcomeFavorable(learn)

playerAgency(cursor)
entityInfluences(cursor,push_idea(cursor))
enables(push_idea(cursor),learn)

playerWillDo(near(cursor,new_idea),push_idea(cursor))

destroys(vectorialist,new_idea)
requiredBy(new_idea,push_idea(cursor))
antagonist(vectorialist)

outcomeBetterThan(producer,initial,forget)
outcomeBetterThan(producer,learn,forget)
```

The system infers that the goal is to maintain `ideasAbsorbed`, that the outcome `forget` affects `ideasAbsorbed` negatively, and the outcome `learn` affects it positively, meaning that `learn` is a favorable outcome. The player has agency over the cursor and uses the cursor to influence the outcome `push_idea(cursor)` which in turn enables the outcome `learn` by pushing ideas towards the producers. As such `learn` is a favorable outcome that the player has indirect control over via pushing ideas with the cursor, so the player will place the cursor near `new_ideas` to try to enact the `learn` outcome. Furthermore, collision with the `vectorialist` deletes `new_ideas` (which we label `destroys`) which are required for a player to influence the outcome `push_idea(cursor)`. An entity not not under the control of the player that destroys an entity required for the player's progress is an `antagonist` (although not the only possible form of antagonism representable), thus the `vectorialist` is an antagonist.

Cultural knowledge about colors (that green represents life and grey a lack of life) leads to the interpretation that the outcome labeled as `forget` leads to a worse outcome for the producer than either its initial state or its state as a result of the outcome `learn`.

Second example

Our next example is Julien Thiennot's *Cookie Clicker*,[4] a resource-driven game that does not rely on spatial movement logics. In *Cookie Clicker,* the goal is to bootstrap automated cookie-production systems that escalate to a constantly-growing tower of upgrades and achievements. The key dynamics in the game are feedback loops and investment—e.g., spending a relatively small amount of cookies to purchase an upgrade leads to the permanent faster production of cookies.

[4] `http://orteil.dashnet.org/cookieclicker/`

Answer Set Programming Encoding

```
% - Clicking cookie increases cookies.
precondition(control_event(click(giant_cookie)), click_cookie).
result(click_cookie, increase(cookies, click_rate)).

% - Time causes cookies to increase by cps.
precondition(tick, produce).
result(produce, increase(cookies, cps)).

% - Buying a producer increases cps by that producer's production rate,
% subtracts cost from number of cookies, & increases the subsequent cost
precondition(control_event(click(button(grandma))), click_cookie).
precondition(ge(cookies, cost), buy_producer(grandma)).
result(buy_producer(grandma), decrease(cookies,cost)).
result(buy_producer(grandma), increase(cost,cost)).
result(buy_producer(grandma), increase(cps, production_rate)).
```

Meaning Derivations As there is no extrinsically-stated goal in *Cookie Clicker*, we have to supply cultural knowledge to the encoding in order to provide a basis for understanding the game. Namely, we add the fact `good(cookies)` to represent the likely player interpretation of cookies as desirable. Below is a subset of the derived dynamics and meanings:

```
hasTradeoff(buy_producer(grandma))

playerWillDo(control_event(click(giant_cookie)),click_cookie)
playerWillDo(control_event(click(buy_button(farm))),buy_producer(farm))
playerWillDo(control_event(click(buy_button(grandma))),buy_producer(grandma))

positive_feedback(cps)

bootstrapping(click_cookie,buy_producer(farm))
bootstrapping(click_cookie,buy_producer(grandma))

investment(buy_producer(farm))
investment(buy_producer(grandma))

lower_cost(initial,buy_producer(grandma),buy_producer(farm)).
allocation_choice(buy_producer(farm),buy_producer(grandma),cookies)
```

The system determines that it costs cookies to increase the cookies-per-second, which then leads to the creation of more cookies, creating a positive feedback loop. Due to the fact that clicking on the cookie is unrestricted, that clicking generates cookies, and that cookies are a restriction on buying grandmas and farms, it deduces that a player must bootstrap by first clicking on cookies to be able to buy a producer. Given the positive feedback loop associated with cookies-per-second and that cookies are good, the system reasons that buying a producer represents an investment. Finally, given that both grandmas and farms require and consume cookies and the initialized values for their costs, it reasons that the player must make a choice of where to allocate their cookies, with grandmas having the lower initial cost of the two (although an indeterminate ordering on an arbitrary time scale).

Other formalizations

In addition to the examples we have described in detail, we have also formalized and derived reasoning about a handful of classic arcade games (*Pong, Kaboom!*) and four games of our own design on the basis of our scenario generation goals. For *Pong,* we derive that the game is a symmetric competition, that the two players are antagonists of each other, and

that each player will try to hit the ball with their respective paddle. For *Kaboom!,* we derive facts such as that the bomber is an antagonist, bombs harm the player, the basket harms bombs, the player will attempt to cause the bombs to collide with the basket, and the player will attempt to avoid causing the bombs to collide with the bottom of the screen. And for our own games, we encode reasoning analogous to design specifications such as "a player must perform a risky hand-eye coordination task to keep their cool" and "the player must make time-sensitive decisions about resource allocation whose consequences have a tradeoff between personal benefit and global cost."

Discussion

We have demonstrated a novel approach to the problem of game specification based on relating a game specification to higher-level meanings. We introduced two novel and distinct knowledge formalisms: the specification language, in which we express game mechanics, and the meaning-level design intent language. The design of our specification language was especially tricky, because it needs to play the dual role of supporting meaning-level interpretation while also unambiguously mapping onto an executable game (making it feasible to write a compiler back-end). The meaning-level design intent language is informed by micro-rhetorics (Treanor et al. 2012b) and proceduralist readings, and it allows for generative variation through a many-to-many relationship: specifications may give rise to multiple meanings, and a single meaning may have many instantiations as a game specification.

Currently, we assess our work as forming a good account of meaning for games with resource logics, movement and collision logics, and graphical communication affordances (including, notably, some high-level representation of continuous physics, which we did not illustrate in our previous examples). Thus, we can already express simple arcade games, platformers, and incremental games in a uniform way, which we assessed would have been very difficult in existing game formalisms such as the Video Game Description Language (VGDL) (Schaul 2013). Furthermore, by formalizing operational logics, we forsee an ability to scale easily to a large range of other common game idioms and perform a similar style of reasoning over them at little extra authoring cost. We plan to expand upon and standardize the Operational Logic Catalog of Osborn and Treanor (Osborn and Treanor 2016) to represent things like inventory systems, card decks, and spatial pattern matching (as in *Tetris* and match-3 games).

Future Work

The long-term goal for this work is to turn the proceduralist reading process on its head and generate games that fulfill a specified reading (playing the role of a design intent) such as *The player must come between the vectorialist and new ideas to prevent commodification*. Even in its current state, the system can generate games given a desired proceduralist reading. The generator effectively runs the interpretation rules "in reverse" to pull together OL pieces, such as collision, controls, or resource transactions, and assign them to preconditions or outcomes subject to the constraint that they will produce the desired reading. However, the generated games currently have too many possible readings; they exhibit a surfeit of meaning, making it difficult for the player to discern the desired reading. Therefore, we are working on adding constraints to rule out games that violate common sense and control the number and complexity of rules and possible interpretations.

As noted in the introduction, this work is intended to be used as part of scenario generator that generates an interactive narrative that is linked to the generated game. Much of our future work hinges on combining these two generators, mediating with a scenario generator that is capable of reasoning about both narrative and mini-game concerns for player experience. As for evaluation, our planned work primarily concerns combined scenarios, but we have determined some possible lines of evaluation for our system as a game analysis engine as well. One design would be to do an expert evaluation, requesting critical readings of games from leading scholars, formalizing their arguments, and comparing the shapes of the derivation trees to the ones our system generates. We could also compare the shapes of generated meaning derivations *internally*, i.e., carry out an expressive range analysis (Smith and Whitehead 2010) over variables such as proof tree width, depth, and number of rules in the reasoning principle base used, repeated, or omitted. Further, we could use these properties of meaning derivations as characteristics of games themselves to reason over or optimize for: perhaps we want games with the fewest possible meanings but whose deepest meaning derivation is as deep as possible, or some other combination of constraints.

In summary, we have presented a novel approach to computational game interpretation based on a formalization of procedural reasoning, resulting in a static analysis relating operational logic-based game definitions to high-level meanings. This work represents the first effort to combine mechanics-level game specification, cultural knowledge, and operational logics in one formalism, resulting in a highly expressive reasoning space.

References

Cook, M., and Colton, S. 2014. Ludus ex machina: Building a 3d game designer that competes alongside humans. In *Proceedings of the 5th International Conference on Computational Creativity*, volume 380.

Hunicke, R.; Leblanc, M.; and Zubek, R. 2004a. Mda: A formal approach to game design and game research. In *In Proceedings of the Challenges in Games AI Workshop, Nineteenth National Conference of Artificial Intelligence*, 1–5. Press.

Hunicke, R.; LeBlanc, M.; and Zubek, R. 2004b. Mda: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, volume 4, 1.

Kuhlmann, G.; Dresner, K.; and Stone, P. 2006. Automatic heuristic construction in a complete general game player. In *Proceedings of the Twenty-First National Conference on Artificial Intelligence*, 1457–62.

Martens, C. 2015. Ceptre: A language for modeling generative interactive systems. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Mateas, M., and Wardrip-Fruin, N. 2009. Defining operational logics.

Nelson, M. J., and Mateas, M. 2007. Towards automated game design. In *AI* IA 2007: Artificial Intelligence and Human-Oriented Computing*. Springer. 626–637.

Nelson, M. J., and Mateas, M. 2009. A requirements analysis for videogame design support tools. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, 137–144. ACM.

Osborn, J., and Treanor, M. 2016. Operational logic catalog.

Salen, K., and Zimmerman, E. 2004. *Rules of play: Game design fundamentals*. MIT press.

Schaul, T. 2013. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, 1–8. IEEE.

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A successful general game player. *Proceedings of the National Conference on Artificial Intelligence* 22(2):1191.

Smith, A. M., and Mateas, M. 2010. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, 273–280. IEEE.

Smith, G., and Whitehead, J. 2010. Analyzing the expressive range of a level generator. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, 4. ACM.

Stolee, K. T., and Fristoe, T. 2011. Expressing computer science concepts through kodu game lab. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, 99–104. ACM.

Treanor, M.; Schweizer, B.; Bogost, I.; and Mateas, M. 2011. Proceduralist readings: How to find meaning in games with graphical logics. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, 115–122. ACM.

Treanor, M.; Blackford, B.; Mateas, M.; and Bogost, I. 2012a. Game-o-matic: Generating videogames that represent ideas. In *Procedural Content Generation Workshop at the Foundations of Digital Games Conference*.

Treanor, M.; Schweizer, B.; Bogost, I.; and Mateas, M. 2012b. The micro-rhetorics of game-o-matic. In *Proceedings of the International Conference on the Foundations of Digital Games*, 18–25. ACM.

Wardrip-Fruin, N. 2005. Playable media and textual instruments. *Dichtung Digital* 34:211–253.