Computational Caricatures: Probing the Game Design Process with AI

Adam M. Smith and **Michael Mateas**
Expressive Intelligence Studio
University of California, Santa Cruz
1165 High Street, Santa Cruz, CA 95064
{amsmith,michaelm}@soe.ucsc.edu

Abstract

We propose the creation of *computational caricatures* as a design research practice that aims to advance understanding of the game design process and to develop the reusable technology for design automation. Computational caricatures capture and exaggerate statements about the game design process in the form of computational systems (i.e. software and hardware). In comparison with empirical interviews of game designers, arguments from established design theory, and the creation of neutral simulations of the design process, computational caricatures provide more direct access to inquiry and insight about design. Further, they tangibly demonstrate architectures and subsystems for a new generation of human-assisting design support systems and adaptive games that embed aspects of automated design in their runtime processes. In this paper, we frame the idea of computational caricature, review several existing design automation prototypes through the lens of caricature, and call for more design research to be done following this practice.

Introduction

Design research generally intends to understand and advance the process of design or to transform the space of artifacts that might result from that process. In many cases, this takes the form of carrying out design projects with larger questions in mind. In this paper, we describe a design research practice which targets game design, specifically addressing questions about the role of artificial intelligence (AI) in the game design process.

In the design of game content artifacts (such as music, level maps, and story fragments), many automated systems draw heavily on AI search techniques (Togelius et al. 2010). In our own game content generation work, we have shown how to use automated inference tools to generate game content from design space captured in a concise knowledge representation (Smith and Mateas 2011a). Earlier, we employed learning and retrieval in a generative art installation (outside of games) that adapted to its audience to stay interesting over multiple months of interaction (Smith et al. 2008).

That the full spectrum of AI has come into contact with automated content generation should be no surprise. The design of artifacts by machines has been a central topic of discussion in computational creativity (Colton, de Mántaras, and Stock 2009), a field that uses theory and system building to advance understanding of both machine and human creativity. Recently, we proposed a connection between several computational creativity and AI topics (artificial curiosity, automated scientific discovery, and knowledge-level modeling) and the full context of creative game design, making predictions about the game design process as carried out by human designers and future machines (Smith and Mateas 2011b).

We believe that following the program of computational creativity in the context of game design will continue to advance our understanding of the design process and unlock the building blocks of a new generation of human-assisting design automation tools and currently-unreachable game systems that embed aspects of the design process in their runtime systems. Thus, the key question to ask is whether a machine can design, and, if so, how? Noted design studies researcher Nigel Cross echoes this sentiment (2006):

> Asking 'Can a machine design?' is an appropriate research strategy, not simply for trying to replace human design by machine design, but for better understanding the cognitive processes of human design activity.

This question can be tentatively answered by many means. Taking an empirical approach, Nelson and Mateas (2009b) interviewed expert game designers and, through playing the role of the interface between a designer and automated reasoning tools, they drew several conclusions about the potential roles for AI in assisting game designers (including accelerating exploratory prototyping, performing early-stage verification of designs, and supporting design-level regression testing). In contrast, our own development of a knowledge-level theory of creativity in game design (Smith and Mateas 2011b) draws weight from established theory in computational creativity, AI, and game design. Distinct from purely-empirical or purely-theoretical approaches, we believe the most convincing answers to the question of whether and how a machine can design games will take the form of computational systems, inspired by theory and constrained by the practicalities of what is implementable with today's computational resources.

We claim that building computational caricatures of the game design process (with all of the subjectivity and bias

that the term implies) will provide direct access to insight regarding the role of AI in the game design process. In this paper, we frame the idea of a computational caricature, review several existing design automation prototypes through the lens of caricature, and call for more design research to be done following this practice.

Computational Caricatures

The practice of building computational caricatures of the game design process is tied strongly to the sense of caricature familiar from visual art, it has precedents in the critical technical practice of AI, and it holds special promise for game design (where aspects of automated design are increasingly being embedded into generative and adaptive games).

Visual Caricature

In visual art, caricature refers to a style of portraiture (creating images of a person with the intent to capture likeness, personality, and mood) in which distortion is used to make the subject more easily identifiable. Caricatures attempt to be a better representation of a subject than an accurate depiction (such as a photograph or photo-realistic painting) would be. Through exaggeration and oversimplification, an artist makes a statement about what is most salient about that subject. Different artists may decide that different aspects of a subject are the most salient or, agreeing on saliency, they may decide to present the same aspects in different ways. As a vehicle for the artist's claims, these loaded portraits can express some ideas more quickly than the equivalent verbal description.

What makes a caricature notable is not just the choice of aspects of the subject which are emphasized but also all of the other choices that go into it. One artist may decide to borrow stylistic elements such as line weight or shading techniques from another when creating future caricatures. In actuality, a caricature captures two kinds of beliefs: "this is what is interesting about the subject", and (implicitly), "this is the most obvious way for it to be recognizably represented".

In the remainder of the paper we will be discussing a kind of abstract caricature which is related to visual caricature through analogy. In this analogy, the subject of a caricature (conventionally a human face) is replaced by a cultural process. In the example systems we describe later, the subject of each caricature is a proposed process for automating some aspects of game design through the use of AI. The medium of the caricature (conventionally ink and airbrush on white paper) is replaced with computational media: software and hardware. The representation strategies used to carry out oversimplification and exaggeration (conventionally enlarging eyes, noses, chins, and ears while eliminating wrinkles and other blemishes) are translated into choices in a computational system's implementation: ignoring certain inputs, focusing on specific subproblems, or placing very strict requirement on potential users of the system.

However popular, the presence of large, airbrushed chins is not the essence of caricature. Instead, it is the general use of oversimplification and exaggeration techniques to make the subtleties of subject more identifiable. As such, in the subsequent discussion of computational caricature, it is not our goal to understand the different ways in which cultural processes can be distorted in order to transform them into software. Instead, we want to highlight the use of caricaturists techniques in a computational medium for the purposes of making deeply technical claims more identifiable.

Procedural Portraits and Computational Caricature

In the general context of using AI in cultural production (which includes the use of AI in the game design process), Mateas (2003a) proposed the idea of building "procedural portraits": representations of human cultural processes in the form of realized computational systems.

Portraits of processes are easily recognizable as simulations. However, simulations greatly vary in the degree to which the simulator agrees with the simulated. ACT-R, a system-as-theory cognitive architecture, has been used to make quantitative predictions about human behavior (Gunzelmann et al. 2011). Such simulations are an attempt at accurate modeling (equivalent to photo-realism in portraiture). By contrast, Weizenbaum's *Eliza*, a simulation of Rogerian psychotherapy in the form of a chatterbot, is best described (in Weizenbaum's own words) as a "parody" (1976). Eliza captures the "nondirectional" nature of Rogerian therapy in a near-stateless program that simply asks the user about the most recent input. Exaggeration and oversimplification are used liberally and openly in a charged simulation like *Eliza* whereas they would be rationalized or explained away in a neutral simulation like ACT-R.

By contrast to general portraiture, caricature allows us to be taken seriously in going after nuggets of truth (or at least proposed truth) without having gotten all of the surrounding details right. One of the values of caricature is rapidity of recognition. The kind of flash-communication afforded by caricatures makes them well suited as conversation-starters. In the context of system building, procedural portraits can make complicated and deeply technical arguments accessible, tangible even. The translation of a (too) familiar human practice into cold, machine crunching can make it unfamiliar enough that we gain the new perspective required to put some old questions to rest and ask more important ones.

In the same way that visual caricature encoded two kinds of beliefs (statements about the subject and statements about representational strategies), caricature in procedural portraits conveys two messages: "this is what is interesting about the human cultural process", and (implicitly), "this is the most obvious way to implement it on a machine". When a particular system architecture provides better affordances for embedding our message than expected, we say that there is "architectural surplus" (Mateas 2003b). Identifying sources of architectural surplus not only improves our ability to communicate through the building of systems, it paves the way for new kinds of systems that use AI to manipulate and create human-appreciable meaning automatically. This process becomes much more concrete when we zoom into the context of game design.

Computational Caricature of the Game Design Process

Computational caricatures are procedural portraits created with the the values of caricature in mind (enhancing recognizability through exaggeration and oversimplification). As such, every computational caricature of the game design process will embed one or more exaggerated and often controversial statements about game design (alternatively thought of as propositions, claims, or hypotheses). Likewise, every computational caricature will make wildly simplifying assumptions in the process of reducing their perspective on design into an arrangement of code that is executable on a machine. The choice of which concerns to abstract away tells us about the intersection of what the caricaturist believes is salient and what is realistically feasible given the implementation techniques known to them. That a particular computational caricature of game design does not address a well-known aspect of human game design practice (perhaps learning through observing human playtesters) does not immediately imply a statement of unimportance. Instead, it might imply that the caricaturist knows of no promising architecture for realizing that practice in their caricature.

The creation of a computational caricature (whether by a game designer, AI scientist, or design studies researcher) will always invite questioning beyond whether the statement the caricature seems to push is simply valid or not. They invite questioning into how accurately the system's knowledge representation models a designer's beliefs, into how the chosen algorithms (perhaps a specific kind of search) captures the designer's working processes, into how the machine's apparent values overlap with or diverge from those in traditional human practice (such as whether the value of a game flows from it's internal structure, from empirical properties only observable in playtests, or some interesting balance of these), and so on.

Tanagra (Smith, Whitehead, and Mateas 2010), a demonstration of mixed-initiative design for platformer levels, seems to invite us to explore the statement "humans and machines should produce game content cooperatively, asynchronously editing a shared design" (our words). Playing with this system suggests follow-up questions: When an infeasible design arises, should it always be the human's responsibility to resolve this? How do we know when a level design is finished, and can machines have an opinion on the subject? Should two humans cooperate to design level in this way? (They currently do not.) The machine seems to be faster at verifying basic playability properties than the human; what other asymmetries are there and how should they be exploited in future design assistance tools?

A visual caricaturist's preference for certain line weights and shading techniques translate into a computational caricaturist's preference for code-level implementation details. Every computational caricature sets, reinforces, or breaks precedents in implementation techniques. Every system (whether the caricaturist is conscious of it or not) makes the implicit suggestion that future systems should use a similar problem formulation, programming language, or software library to a given concern.

Even the details of a caricature that are invisible without deep inspection become potential foundations for future system and theories. In a review of content generation systems based on answer set programming that involved a code-level analysis (Smith and Mateas 2011a), we found that every system assembled fragments of logic programs on the fly in a process of "dynamic program construction". Dynamic program construction, an architectural motif that appeared to its first users as an implementation detail, has emerged as a standard practice that future declarative, solver-based content generators should employ. In an interpretation of logic program fragments as encodings of a designer's beliefs about a design space, dynamic program construction implies a computational model of designers that heavily recycle domain knowledge between the analysis and synthesis phases of design.

Taken together, the building of computational caricatures is a design research practice that, while advancing personal goals (such as sharing opinionated claims about game design and the technology that should power future design automation systems), is centered on rapidly communicating deeply technical statements about game design and the role AI can play in its process. By building and sharing these systems, the caricaturist simultaneously accelerates the discourse around game design and produces tangible products of value along the way: systems which engage in a human cultural process. The reader should not be convinced by our claims alone, however. The best demonstrations of the value of the practice of computational caricature are the caricatures themselves.

Exemplars

In this section we review three (of many) examples of computational caricatures of the game design process found in the wild. None of these systems were designed explicitly as computational caricatures; nevertheless, it is possible to pick out statements about design that each system seems to be pushing and note where unintended details have also lead to revelations about game design. While many of the systems are the product of joint work, we attempt to identify the caricaturist behind each system and speculate on their individual motivations. A sketch of how each of these examples works as a computational caricature can be seen in Table 1.

A Designer in a Box

Cameron Browne's Ludi is a board game designer in a box, or in Browne's words "a system for playing, measuring and synthesizing games" (Browne and Maire 2010). In its game synthesizing subsystem, Ludi uses a genetic algorithm to search the space of games expressible in the same language understood by its (also search-based) game playing subsystem. Ludi judges the value of potential games based on properties of typical playthroughs (records of simulated play between modeled players). These game properties are encodings of natural-language concepts like "completion", "duration", and "uncertainty" in a mathematical formulation.

Ludi seems to answer our question of whether a machine can design with resounding "yes" and continues with

the claim that "machines are human-competitive designers". *Yavalath* is a grid-based strategy game designed by Ludi (named by Ludi as well) that is commercially available. On the popular site BoardGameGeek, *Yavalath* ranks[1] just above the puzzle card game *Set*.

In reaching directly for the machinic construction of a human-appreciable game, Ludi is blissfully unaware of the potential feedback from the human playtesters (it does not employ any), the design patterns used by any of the games outside of its very specific niche genre, and the need to redefine one's own representation system to eventually express new kinds of artifacts. These are not shortcomings of the system; instead they are the simplifications that made possible transforming Ludi from a thought experiment into a live system. The physical existence of Ludi and its product, *Yavalath*, materially changes the conversation around machine design: machines demonstrably *can* design real, valuable games, and what remains now is the question of how machines *should* design (why machines design, how we should assign credit/blame between machine and programmer, and the multitude of related open questions).

Though the particular properties selected for use in Ludi's game evaluation routine are presumably not part of the caricaturist's intended claim, Ludi demonstrates that there are mathematical properties that we should look for in two-player, strategic board games that do not immediately reduce to classical game theory. This is an unexpected result for human game design that also suggests a role for machines in the design of future board games (as verifiers and automated explorers of localized design spaces).

For machine design, *Yavalath*'s terse construction in Ludi's game description language raises interesting questions about how much the designer of a representation language affects the products of the systems that use it. Did Browne, who is independently an experienced board game designer, do the hard work behind inventing *Yavalath* by pre-selecting a representation that was rich with interesting game designs? Similar issues were raised by the by the reinvention of fragments of set theory by the mathematical discovery system AM (Lenat and Brown 1983), resulting in a discussion that has shaped the discourse around automated discovery for several decades.

Cooperating with the Machine

Gillian Smith's Tanagra is "a prototype mixed-initiative design tool for 2D platformer level design, in which a human and computer work together to produce a level". Upon starting, Tanagra presents the user (assumed to be a level designer) with a blank canvas and the basic ability to paint platforms into the game world. A large button labeled "Start Generator" is also present and, when pressed, results in the near instantaneous filling of the canvas with familiar platforming elements (platforms with gaps, enemies and stompers), all placed to conform to the systems internal rhythm-inspired design theory and the limits placed on the player's avatar by the associated game's mechanics. With functionality in place for both completely unassisted human level design and completely automated machinic level design, Tanagra invites us to reflect on the give-and-take between the two designers at work. In a typical demonstration, the human operator will draw only a few sparse platforms, place high level requirements on large gaps in the partial design to be filled, and perform minor aesthetic clean-up activities before declaring the level completed (leaving nearly all of the low level platform sizing, placement, splitting and recombining to the machine).

Just a few minutes of observing Tanagra interacting with a human user (level designer or otherwise) begins to raise the questions mentioned in the previous section. The idea of mixed-initiative design for geometric artifacts where the human operator expresses high level constraints and leaves the machine to adjust the details is hardly new, perhaps originating in Sutherland's *Sketchpad* system (1963). Nonetheless, the existence of Tanagra, operating in the more familiar domain of platformer level design transforms the abstract discussion about the distant potential of a mixed-initiative setup in future game design tools into a more concrete discussion. We can now ask very specific, technical questions inspired by Tanagra's implementation: Why do no commonly used level design tools have any support for expressing design constraints (such as reachability of some location by the player's avatar), even without the ability to automatically satisfy them?

Where Tanagra zooms ahead of the industry standard platformer level design tools in support for intelligent assistance (perhaps providing more support than is needed as an exaggeration to ease one's recognition of the system's aims), it sharply oversimplifies other aspects usually required of platformer level design tools: exporting files for use in an external game (Tanagra lacks a "save" button), importing custom tilesets, and placing additional level details such as background art, flying enemies, optional paths, etc. As a computational caricature, these distortions of platformer level designed are welcomed in exchange for a tangible, interactive demonstration of a potential future for level design tools and a validation of the software architecture that made it possible.

On the implementation side, Tanagra is the composition of a reactive planner and a numerical constraint solver (an unprecedented choice in level design tools, to say the least). The caricaturist suggests that reactive planning is a useful top-level architecture for managing the mixed-initiative interaction and orchestrating the high-level search processes involved in geometry generation. However, she does not attempt to characterize level design as a constraint solving process (nor is the integrated constraint solver used, architecturally, as anything more than a supporting software library). That constraint solving (through answer set programming) played a key role in several content generation systems (Smith and Mateas 2011a) suggests that the use of constraint solvers is a source of architectural surplus. That this surplus can be attributed generally to constraint solvers (which take a declarative specification of a problem's design concerns and produce a satisfying assignment of numerical and structural properties) and not specifically answer set

[1] `http://boardgamegeek.com/boardgame/33767/yavalath`

solvers is a direct result of Tanagra's seemingly incidental use of a numerical constraint solver. Given the leverage provided by constraint solvers, we are inspired to think of alternate formalizations of game design that foreground the creation and satisfaction of constraints.

Imagining Gameplay

Adam Smith's (the first author's) Ludocore is a "logical game engine" for producing queryable, logical models of the core rule systems of a game (Smith, Nelson, and Mateas 2010). Many game engines intend to ease implementation of complex videogames by abstracting away the details of 3D rendering, asynchronous resource loading, and many other technical challenges. By contrast (and through extreme exaggeration), Ludocore is intended to ease implementing games that lack not only graphics and sound but also any form of live player input. Instead, Ludocore abstracts a videogame down to the central rules that govern the primary game state and how that state is affected over time by game events.

Using Ludocore, a designer can rapidly encode and iterate on the design of the most formal aspects of their game's design. In exchange for hyper-formalization, Ludocore promises the ability to imagine gameplay for these as-yet incomplete games. The system's "gameplay trace inference" affordance allows a designer to ask for a symbolic gameplay trace, from the vast space of potential low-level action sequences possible in a game, that satisfies arbitrary logical constraints. In this representation, questions such as "is the game winnable?", "how would someone get from here to there without using this special item", and even "how should I connect the various regions in my level design such that they player cannot escape without encountering all of my content?" all reduce to a common representation.

Through exaggeration and oversimplification, Ludocore manages to realize an interactive prototype of a system that can imagine play for arbitrary games. This capability, of course, is conditioned on the "designer" being an experienced logic programmer, the rules of the game being primarily symbolic as opposed to numerical, the requested properties of gameplay being expressible in a subset of first-order logic, and having to wait unreasonable lengths of time for the results of certain kinds of queries. It is on top of this admittedly shaky foundation that Ludocore makes it's statements about AI in the game design process: deeply modeling videogames requires capturing not just the game's rules, but also the configuration of the game's world, a body of assumptions about the kinds of players who might play the game, yet another body of assumptions about the situation in play that currently interests the designer, and, on top of that, the ability to reason through all of this to generate concrete gameplay traces which tell the designer something that was out of range of their human inferential ability. In other words: deep computational modeling of gameplay is hard, but possible.

Ludocore's method of inference (using an answer set solver) is based on a system of exhaustive search. While this procedure for reasoning is alien to us (it is a poor model of how designers actually imagine gameplay traces), it does lead to example gameplay traces which we would not likely think of ourselves. Relatedly, Ludocore's hyper-declarative programming language (which recycles Prolog syntax) is highly unfamiliar to most game designers (who are more likely to be familiar with an imperative programming paradigm, if any). Nonetheless, this logical representation seems to be particularly well suited for use by machines, and it has seen reuse in other design automation prototypes (Nelson and Mateas 2008).

Where Ludocore is the most extreme instance of computational caricature that we review here (with the least generally accessible results), such extreme distortion allows for demonstrating interactions that would be entirely unreasonable in a neutral simulation of design processes. The more promising facets of an extreme caricature can be recycled into and evaluated in the context of more tame caricatures. For example, the "structural query" feature of Ludocore (which produces static world configurations as a result instead of traces of dynamic gameplay) was isolated and extracted as the less complicated practice of simply using answer set solvers to power game content generators (mentioned above). Ludocore's knowledge representation (but not inference techniques) were recycled in the Biped game prototyping tool which added graphics, sound, and (most importantly) live player interaction with early-stage game prototypes (Nelson and Mateas 2009a).

Summary

As a caricature, each of the example systems attempts to make some technical claim about AI in the game design process easily recognizable. In doing this, it will make many simplifying assumptions, some of which are oversimplifications of the design process that should be overlooked while others are promising abstractions to be reused in future computational systems (be they subsequent caricatures or future design automation and game systems). Table 1 summarizes each of the above system's status as a caricature with a list of claims, oversimplifications, and abstractions.

Conclusion

Towards the goal of better understanding the role of AI in the game design process (both for what this tells us about human game design and future design assisted by and embedded in machines), we have described how creating computational caricatures accelerates discourse and uncovers promising implementation techniques which exhibit architectural surplus. These caricatures (rich with subjective bias, exaggeration, and oversimplification) provide more direct inquiry into the questions of if and how machines can design than would more neutral simulations of the design process. Further, they appear to be more effective at exploring deeply technical ideas about the design process than do purely empirical or purely theoretical approaches.

We hope to inspire the creation of many more computational caricatures of the game design process, and we welcome exaggeration and oversimplification in the service of transforming distant ideas into tangible systems. We intend that this design research practice produce valuable results for AI, game design, and design studies generally.

Table 1: A (non-exhaustive) summary of the example systems' claims about and oversimplifications and abstractions of the role of AI in the game design process.

System	Claims (to be quickly recognized)	Oversimplifications (to be overlooked)	Abstractions (to be reused in the future)
Ludi	Machines can automatically design games that humans genuinely appreciate.	• Ruleset invention boils down to sampling from a given, genre-specific grammar. • Games can be evaluated without any human player interaction.	• Using simulated play to evaluate candidate rulesets • Quickly rejecting potential designs with easy to detect flaws
Tangra	Humans and machines should collaborate by asynchronously modifying a shared design.	• Produced levels need never be extracted from the tool nor imported into a separate game. • Two designers can sufficiently communicate through design edits alone.	• Using constraint solvers to quickly resolve low-level design problems • Enforcing limits imposed by the game's mechanics during level design
Ludocore	Beyond worlds and rule systems, designers amass assumptions about players and play.	• Designers think like SAT solvers, and they read/write logic programs. • Arbitrary game rules are naturally expressed in declarative logic formalism.	• Using pre-existing solvers to automatically generate content and imagine gameplay traces • Quickly capturing the high-level, symbolic mechanics of a game in a small amount of declarative code

Acknowledgments

This work was supported in part by the National Science Foundation, grant IIS-1048385. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

Browne, C., and Maire, F. 2010. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games* 2(1).

Colton, S.; de Mántaras, R. L.; and Stock, O. 2009. Computational creativity: Coming of age. *AI Magazine* 30(3):11–14.

Cross, N. 2006. *Designerly Ways of Knowing*. Springer.

Gunzelmann, G.; Moore, L. R.; Salvucci, D.; and Gluck, K. A. 2011. Sleep loss and driver performance: Quantitative predictions with zero free parameters. *Cognitive Systems Research* 12(2):154–163.

Lenat, D. B., and Brown, J. S. 1983. Why AM and Eurisko appear to work. In *Proceedings of the National Conference on Artificial Intelligence (AAAI-83)*, 236–240.

Mateas, M. 2003a. Expressive AI. In *Electronic Art and Animation Catalog, Art and Culture Papers, SigGraph 2000*.

Mateas, M. 2003b. Expressive AI: A semiotic analyis of machinic affordances. In *Proceedings of the 3rd Conference on Computational Semiotics and New Media (COSIGN 2003)*.

Nelson, M. J., and Mateas, M. 2008. Recombinable game mechanics for automated design support. In *Proceedings of the 4th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2008)*.

Nelson, A. M. S. M. J., and Mateas, M. 2009a. Computational support for play testing game sketches. In *Proceedings of the 5th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2009)*.

Nelson, M. J., and Mateas, M. 2009b. A requirements analysis for videogame design support tools. In *Proc. 4th Intl. Conf. on the Foundations of Digital Games (FDG)*.

Smith, A., and Mateas, M. 2011a. Answer set programming for procedural content generation: A design space approach. *IEEE Transactions on Computational Intelligence and AI in Games* preprint.

Smith, A. M., and Mateas, M. 2011b. Knowledge-level creativity in game design. In *Proc. of the 2nd International Conference in Computational Creativity (ICCC 2011)*.

Smith, A.; Romero, M.; Pousman, Z.; and Mateas, M. 2008. Tableau machine: A creative alien presence. In *AAAI Spring Symposium on Creative Intelligent Systems*.

Smith, A. M.; Nelson, M. J.; and Mateas, M. 2010. Ludocore: A logical game engine for modeling videogames. In *IEEE Conference on Computational Intelligence and Games (CIG 2010)*.

Smith, G.; Whitehead, J.; and Mateas, M. 2010. Tanagra: a mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, 209–216.

Sutherland, I. E. 1963. Sketchpad: A man-machine graphical communication system.

Togelius, J.; Yannakakis, G. N.; Stanley, K. O.; and Browne, C. 2010. Search-based procedural content generation. In *Proceedings of the EvoStar Conference*. Springer-Verlag.

Weizenbaum, J. 1976. *Computer Power and Human Reason: from Judgment to Calculation*. W. H. Freeman and Company.