A Rogue Dream: Automatically Generating Meaningful Content For Games

Michael Cook and Simon Colton
Computational Creativity Group
Goldsmiths College, University of London
ccg.doc.gold.ac.uk

Abstract

Procedural content generation is often seen simply as a means to generate '*stuff*', elaborating on or rearranging abstract data types that describe levels or modular pieces of gameplay. Generating content which is situated in an understanding of the real-world is a much harder task; it requires access to large amounts of knowledge, and a good technique for parsing and using that knowledge. In this paper we describe *A Rogue Dream*, a game prototype which can generate new visual content and change its design based on an input word from the player at the start of the game. We describe the game and the tools it makes use of to do this, and use the game to discuss ways in which such techniques might enable unique kinds of gameplay or new directions for intelligent design tools.

Introduction

Procedural content generation (PCG) is commonly used in almost every kind of game development, from experimental self-expression (Kopas 2014) through to AAA mainstream production (Studios 2011), and is a common theme in contemporary games research (Togelius et al. 2011). These PCG systems, almost without exception, have no awareness of the real-world context used by the game in which they are situated, however. Civilisation's map generator does not know about the politics of war for scarce resources, for example. Rogue's dungeon generator has no knowledge of Tolkien lore. For the most part, PCG is used to generate content which can be designed independent of context - the generator operates on abstract data structures that are not dependent on real-world knowledge or cultural understanding.

This status quo is understandable, given that the context surrounding a game is typically where the meaning or atmosphere of the game is conveyed, something that a designer may have the strongest attachment to and thus is least willing to cede to a generator. A game about being an Italian plumber might not have an artistic message to tell the player *per se*, but the feeling of being an adventurous hero is clearly something derived from the designer's own interests and desires as an artist, and integral to the playfulness of the game. For many developers, whether aiming to create a blockbuster hit or simply expressing the feelings of a single person, the game's meaning and context is perhaps the last thing they would wish to give software control of. On the whole, PCG is largely reserved for dealing with issues of scale or replayability – a thousand levels is better than a hundred, a thousand weapons is more unpredictable than ten.

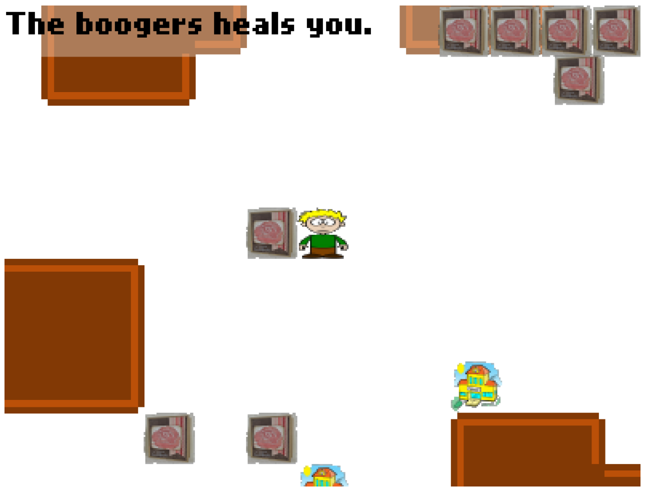


Figure 1: A screenshot from *A Rogue Dream*. The input noun is 'Kid'. The player must avoid roaming School enemies, and can collect Booger items to gain health.

In this paper we will describe a PCG system which allows for dynamic reskinning of a game, changing its surface-level context in response to input words. This system is not limited to a dictionary, or a prepared catalogue of possibilities, but will attempt to skin a game based on any input phrase, sometimes unsuccessfully. While it is common to think of a game's message as integral to a game's human touch, we believe that the application of PCG to this problem may reveal new kinds of game experience, and new elaborations on existing genres. It may also help solve a larger and more longer-term problem: that of how to give an automated game designer the capacity to connect real-world concepts to its games.

We describe here *A Rogue Dream* (ARD), a *roguelite* in which the game's theme and context are decided upon at runtime in response to player input. The game is able to change its characters and enemies, rename and redesign player abilities, and could soon be able to convey simple meanings through the way in which it reskins itself. The remainder of this paper is organised as follows: in the section *A Rogue Dream* we give details of the tools and libraries it accesses in order to dynamically restructure the game. We also give examples of skinnings of the game and how the final game is structured as a result. In *Discussion* we discuss the potential of automating the act of designing new themes for a game at runtime; finally in *Future Work* we look to extensions of the system, and offer conclusions.

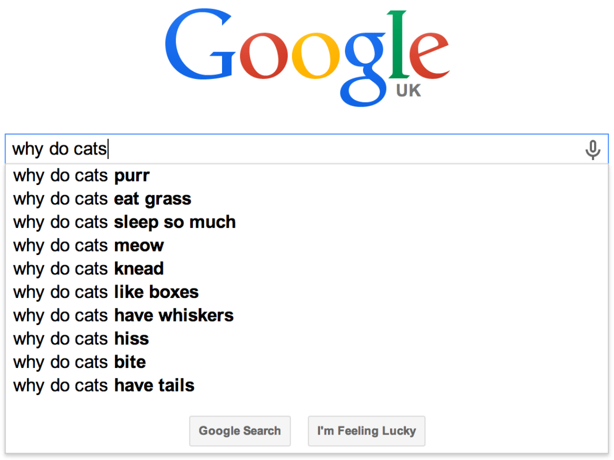
A Rogue Dream

A Rogue Dream (ARD) is an in-development *roguelite*, a roguelike game which drops many of the more intensive requirements of the genre while retaining some of the overall structure. This is the second version of the game to date, reimplementing the original version which was developed during a game jam (Cook 2013a). ARD is a turn-based 2D game played on a square grid in which the player must find a level exit while avoiding or fighting enemies, and collecting optional health pickups. A screenshot of the game is shown in Figure 1. At the beginning of the game, the player is presented with a text box which reads '*Last night, I dreamt I was a...*'. The player is then prompted to enter a noun to complete the sentence. The game then begins loading and skinning the game based on the input, assuming that the player is taking the role of the given noun. The remainder of this section steps through the same processes the game does prior to play commencing.

Data Gathering

A Rogue Dream relies on a technique called *Google Milking* in order to gather information on the input noun from the player. This technique is initially described in (Veale 2012), where it was used to gather data for a linguistic tool called Metaphor Magnet. The technique relies on the use of Google Autocomplete results to mine data from the web. It assumes that when a question is asked of Google, such as *'Why do doctors wear white coats?'*, the asker believes the statement contained in the question (in this case, the asker believes that doctors wear white coats). By asking Google partial questions such as *'Why do doctors'*, Google will suggest endings to the queries based on commonly entered search terms. The endings to these questions can be extracted and stored as usable knowledge. Figure 2 shows a screenshot of Google's autocompletion of *'Why do cats'*.

ARD constructs a normal Google milking query by inserting the target noun into the phrase *'why do [noun]'*. ARD attempts to pluralise the noun but currently we do not employ intelligent tools for doing this accurately in all cases. ARD then extends the standard query by adding keywords after the initial phrase to augment the search. We use several different keywords for gathering different kinds of information. Currently we search for four different pieces of information: enemies, items, goals and abilities. The keywords appended to the searches are shown in Table 3, except in the case of abilities, which we will cover later.



why do cats
why do cats **purr**
why do cats **eat grass**
why do cats **sleep so much**
why do cats **meow**
why do cats **knead**
why do cats **like boxes**
why do cats **have whiskers**
why do cats **hiss**
why do cats **bite**
why do cats **have tails**

Figure 2: Google autocompletion results for questions about cats. (Google and the Google logo are registered trademarks of Google Inc., used with permission.)

Data Type	Keywords
Enemies	'hate', 'hurt', 'destroy'
Items	'have','wear','eat'
Goals	'need','love','like'
Abilities	N/A

Figure 3: Keywords used to augment Google searches.

The results of these searches are sorted according to the type of game content they relate to, so that the Enemies category, for example, contains the results of all three enemy-related keyword searches. In the event that no results are found for a particular data type the game can insert a random noun, remove the entity type from the game entirely, or simply ask the player for another query. Currently we use the latter option and request another noun from the player, although because of the time involved in processing a single query, it may be better for future player experience to use other more robust solutions such as replacing it with a random noun from a predefined list.

Once ARD has gathered data for each data type it then chooses one from each list to be included in the current game, randomly. A point of future work is to make this process more intelligent so as to choose results which can be elaborated on with further PCG systems. We discuss this in the Future Work section below.

When the targets have been chosen for each game entity type, such as enemies or items, ARD then uses Spritely to generate images to use in-game. An example set of results for 'cat' would set the enemy to be 'water', the collectible items to be 'grass' and the goal to be a 'box'.

Spritely

Spritely is a Java-based tool for creating small-scale sprites for use as placeholders in game projects. The tool was released by the author to help produce placeholder graphics

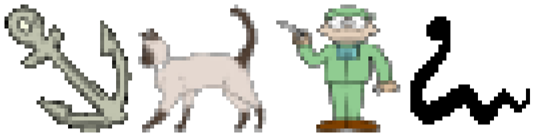


Figure 4: Output from Spritely. Left to right: Anchor, Cat, Doctor, Snake. Outputs not recoloured. 32×32 resolution.

for Ludum Dare entries (Cook 2013b) but the original motivation for the tool was to allow for the generation of graphics at runtime in a game, allowing visual content to be created dynamically.

Spritely takes as input a query string which it uses to search several image repositories for results. In addition, it can also take parameters to affect many aspects of its execution. Currently, the repositories Spritely queries are Google Images, Open Clipart and Wikimedia Commons. When searching these databases, Spritely will also perform other augmented searches to improve the quality of search results. For example, when searching Google Images for a search string *S*, Spritely will also search for *"cartoon S"* and *"silhouette S"*. These additional searches tend to return images which are more likely to pass the filtering process, described below, as well as finding images which look better when scaled down to small resolutions.

When Spritely has retrieved a collection of images, it then filters for images which are suitable for conversion to sprites. To do this, it identifies images which have large areas of similar colour around the entirety of their borders. Areas of colour like this are easy to extract automatically, and by ensuring it covers the entire border of the image we decrease the likelihood that the image is cropped in some way. Once Spritely has filtered the images for ones with clean borders it can then remove these areas, leaving a central image surrounded by transparency. We then downscale the image, without anti-aliasing, to 16×16 resolution and save the result. Spritely has many optional parameters, including disabling certain repositories, randomly selecting images, colourising images using palettes scraped from Colour Lovers[1] and changing the output resolution. Figure 4 shows some hand-curated example output from the system.

ARD retrieves one image for each game entity: currently the enemy, item pickup, player character and the goal. It then stores these locally and loads them into the game once the generation process is complete.

Abilities

In the current version of ARD, there is one final task prior to play – the player is assigned a bonus ability they can activate with the spacebar. An open problem in automated game design is being able to generate connections between arbitrary real-world concepts and game mechanics (Cook and Colton 2013). Later in this paper we discuss a desire to generate abilities that more closely link to the current player character. Currently, ARD looks for key verbs in the Google results and connects these to pre-made character abilities. For example, any of the words 'shoot', 'throw' or 'fire' cause ARD to generate a ranged attack that can destroy enemies from a distance. Other abilities can heal the player if they take damage, or teleport them instantly around the map. In the event that no ability match can be found, the game defaults to a randomly-chosen mechanic, with a word randomly taken from the ability list compiled in the data-gathering stage.

Gameplay

The game loop in ARD is very simple at the moment – the game can largely be considered a prototype, although we explore possibilities for using our approach in games in the later Discussion section. The player controls a character modelled around the noun they supplied at the start of the game. They have four life points, represented by icons at the top of the screen. They can regain health lost in combat by collecting items with the same icon strewn across the map, but their health cannot be raised beyond the starting state of four points. Enemies move randomly around the map, represented by the icons generated from Spritely.

The player must navigate the game's level, looking for an exit object. They can attack enemies by moving into them, but can also be attacked in return. By avoiding combat, or fighting and collecting health, the player explores until the exit is discovered.

Example Skinnings

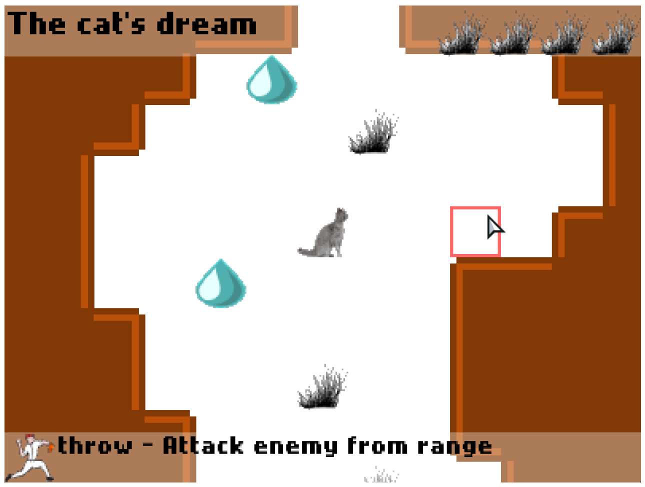
Figures 5a, 5b and 5c show screenshots from A Rogue Dream with three different input words. In this section we will briefly describe each one to give an idea of the game produced. Figure 5a was generated from the input word 'cat'. The enemies of the cat are water droplets, their health is represented by long grass, and their goal is a cardboard box. Figure 5b was generated from the input 'kid'. The enemies of the kid are schools, their health is represented by friends, and their goal is cinnamon toast crunch. Figure 5c was generated from the input 'musician'. The enemies of the musician are pictures of the artist Kenny G, their health is represented by long hair, and their goal is to reach some drugs.

Notes On Success Rate

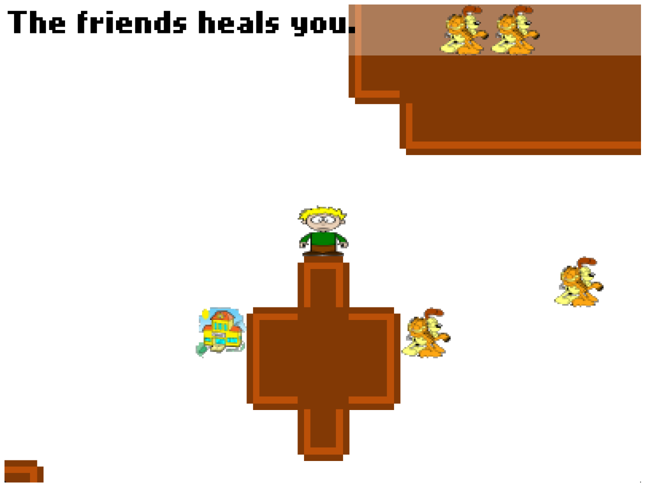
A Rogue Dream's reliance on volatile, noisy data means that it does not always find results, and even where it does find results they do not always make sense. To give an indication of its effectiveness, we performed a simple experimental run of the system on 30 words, picked from three Top Ten lists of animals, jobs and countries[2]. 60% of the words resulted in a full skinning of the game, while 96% of the words resulted in all but one game element being properly skinned. In terms of quality, we performed a curation coefficient analysis as described in (Colton and Wiggins 2013) – we measured what proportion of the results we would be willing to show to someone as quality output of the system. We considered 66% of the results to be good and worthy of showing others. The remaining third were let down either by bugs in

[1]http://colourlovers.com/

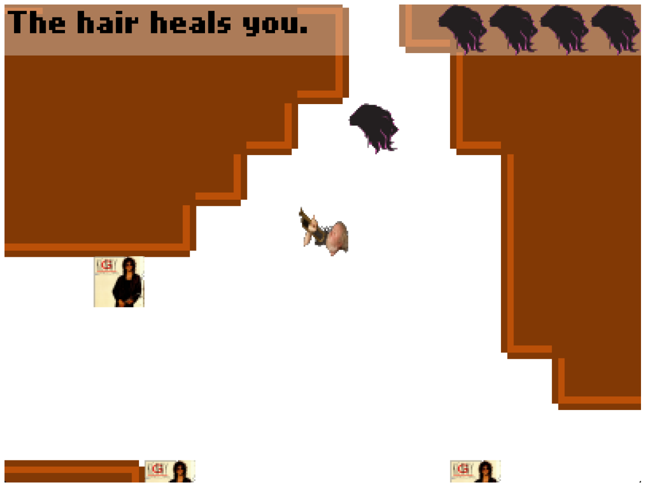
[2]http://www.thetoptens.com/lists/



(a) A screenshot from ARD with the input '*cat*'



(b) A screenshot from ARD with the input '*kid*'



(c) A screenshot from ARD with the input '*musician*'

the system, misinterpretation of terms through Spritely, or through bizarre or unusual output from Google.

It should be noted that the three categories we chose - jobs, animals and nations - are all things to which sentience can be ascribed to, or are spoken in terms of being sentient. This affects the nature of Google results significantly. A Rogue Dream certainly doesn't work on all words, or even all nouns. Nevertheless we consider it an interesting starting point for further expansion.

Discussion

A Rogue Dream as a Mechanic

The original version of A Rogue Dream was described by one player as 'like playing a videogame against The Internet'. The reason for specifically likening it to the Internet, rather than real-world knowledge, likely stems from the fact that the information scraped by ARD is often tainted by popular belief, misconception, stereotype and prejudice, as opposed to purely factual information. For example, if the player inputs *man* in the original build of ARD, the game gives the player the ability to *lie* and *cheat*. In addition to being unreliable, the information ARD gains from the Internet is also highly volatile. In early 2013 when the original version of ARD was built, the input *journalist* would include Syria as an enemy. At the time of writing, in mid-2014, the same query lists newsrooms as the enemy instead. Because the information is dependent on what queries people are typing into Google, the data shifts in line with popular culture, world events and public opinion.

While the current implementation of ARD uses the input word primarily as a means of reassigning the game's theme, we can imagine ways in which this could be extended so that the game's theme, and the choice of input word by the player, becomes intertwined with the game's mechanics. Consider a new version of ARD in which some elements of the game world are already fixed at the beginning of the game, and the player can change character at will by selecting new words for ARD to use. By switching between characters, the player can overcome puzzles and combat encounters set by the game. Suppose the player is controlling a child and comes across a wall of schools – the player is now challenged to think of something that the Internet might agree would like schools or be able to bypass them (a teacher or parent perhaps).

When interacting with ARD, the main attraction for players seems to be in selecting a word and seeing the relationships and phrases that come out of their choice. By making this process part of gameplay, the use of live data from Google or other Internet sources becomes more than just window dressing, and we can explore popular culture – and potentially other data sets like social networks – through games like ARD. We discuss some possible extensions to ARD in Future Work, but the use of data from the Internet, gathered at runtime, is an underexplored but rich area for game design, and the volatility and breadth of the data poses many interesting challenges for artificial intelligence research in terms of interpreting, filtering and applying the data to games in meaningful and intelligent ways.

A Rogue Dream as a Design Tool

Intelligent design tools promise to fundamentally improve the way games are designed, helping developers plan out game designs (Butler et al. 2013), or improve them in real time (Liapis, Yannakakis, and Togelius 2013) (Smith, Whitehead, and Mateas 2010). A Rogue Dream's ability to produce new entities which connect to existing ideas could make it a casual design assistant capable of proposing inspirational ideas, even if it isn't able to understand the entirety of a game's context. We can imagine a new version of A Rogue Dream in which the player can design levels and place objects in the game world. As they do so, A Rogue Dream generates suggestions for new entities by performing searches on the objects that the designer has already added to the game world.

Results could appear as popups at the side of the screen, that the player could either accept and integrate into their game, or edit and change before using (or simply reject). Although the level of sophistication in ARD's suggestions would be quite low currently, it would still serve as a source of inspiration, particularly for younger designers. Because the tool would be co-creative with a person taking primary control of the resulting artefact, ARD can offer more suggestions than normal, because it doesn't need to filter its output to the same high level of confidence. For example, it might offer a list of suggested abilities for a character. In the prototype version of ARD presented here the system has to be very sure that an ability makes sense before using it, and so the range of abilities is very limited. However, if the system is there mainly to serve as inspiration, it can suggest more options and allow the designer to fill in the blanks and integrate it into the game properly.

With a wider corpus of understood keywords (such as the 'hates' or 'loves' indicators that filter search queries), A Rogue Dream could also reinterpret existing designs and suggest new representations for them, in a similar manner to how (Treanor et al. 2012) can interpret an initial graph of game entities in different ways. For example, a designer may have placed the player in the role of a kid, who avoids bullies and tries to find his parents. ARD might suggest that this could be swapped around, putting the player in place of the bully and chasing after the kid. ARD can then fill in new parts of the design with suggestions based on the player as a bully – ARD currently suggests that bullies hate nerds, so they can replace the enemy types from the original design.

Future Work

We discussed long-term goals for the ideas ARD represents in the Discussion section above. In this section we discuss shorter-term goals for ARD specifically, in order to make the game more playable and to elaborate on its strengths.

POS Tagging and Other Linguistic Tools A major limitation in ARD's ability to execute and filter requests to Google is its lack of linguistic knowledge. Assessing which kinds of words are present in the autocompletion results would allow it to better identify activities, items and perceived properties of its subject matter. Embedding a part-of-speech tagger into ARD would let it distinguish verbs from adjectives and so on, allowing for more fine-grained decision-making based on the output of the search.

ARD could also benefit from the integration of other simple language tools such as the ability to pluralise arbitrary words, something which was present in the earlier version of the game but was not ported to the new platform. Language tools like this allow for both the input word and the found phrases to be manipulated and used in new searches. This is particularly important because the kinds of autocompletion we are interested in normally pluralise words (e.g. 'why do *cats* hate *dogs*') but when used in a game the singular form is normally preferred (e.g. 'You scratch the *dog*'). The cat and dog examples are trivial, but many words are irregular, and conjugating verbs for use in queries would be even more complex to do without extra software support.

Metaphor Magnet Representation of the input word and related concepts in ARD is currently undertaken in a literal sense – for example, the player is a soldier, Google suggests that soldiers kill civilians, so the soldier chases after civilian entities in the game to destroy them. A future direction for the game is to allow it to interpret the initial player input in a more metaphorical sense, using Metaphor Magnet (Veale 2012) and ConceptNet (Liu and Singh 2004) to construct new relationship graphs that map those detected through Twitter onto existing relationships in known knowledge bases.

Metaphor Magnet, initially described in (Veale 2012), can suggest metaphorical vehicles to convey certain concepts or ideas. For example, one possible result for *soldier* in Metaphor Magnet is *wolf* Suppose we wish to use this metaphor instead of the original soldier entity. In order to do this, ARD needs to find a second entity that can take the place of civilians in the original relationship (soldier kills civilian). This second entity would need to have a similar relationship to wolf. We can use further Google milking to find such entities ('*why do wolves kill sheep*' is one such result), and then try to construct a game that represents the original player request indirectly, instead of explicitly using the input theme.

Metaphor-based generation might also allow for ARD to add twists on the results it finds through Google, if we can understand the requests well enough. For example, racists are like vicious snakes, according to Metaphor Magnet. Instead of simply replacing a picture of a racist with a picture of a snake, we can use this information to extend the metaphor into the game, perhaps by giving the snakes a poisonous venom attack that hurts people. This extends the metaphor (by making the player think about what the venom might represent) and makes it more than just visual replacement (by linking it to the gameplay).

Conclusions

In this paper we described A Rogue Dream, a simple roguelite game that can dynamically reskin parts of its design in response to player input, using data from the Internet. Unlike contemporary work operating on surface-level context such as (Treanor et al. 2012) or (Nelson and Mateas 2007), ARD does not rely on static, filtered corpora such as

ConceptNet, and as such has a broader scope for possible skinnings, but is subsequently less reliable and more noisy. We described the techniques and libraries it uses to perform its reskinnings, as well as discussing the main game loop and showing some screenshots from possible game inputs; we discussed the possible extensions of the technique that would enable new kinds of game interaction and possible benefits for intelligent design tools; and we looked at ways in which A Rogue Dream could be extended through future work.

Procedural content generation as a primary mechanical force is underexplored in games. A Rogue Dream's use of data from the web makes the game unusually dynamic, interesting to explore and probe, and while the game is simplistic, we believe it shines a light on some possibilities that procedural content generation might offer. However, the tradeoff of working with such rapidly changing and noisy data is that using such data coherently and reliably is very difficult. A Rogue Dream is often broken or confusing for many inputs, and this poses many challenges for AI research in the future, but these challenges may have important results beyond just a few games, and may help address larger issues in automated game design and procedural content generation.

Acknowledgements

The authors wish to thank Antonios Liapis, Azalea Raad and Tony Veale for helpful discussions and suggestions between versions of A Rogue Dream. The reviewers all gave very candid and encouraging feedback and we did our best to incorporate some changes to the paper, including an expanded results section - thank you all for your reviews. This work was funded by EPSRC grant EP/L00206X/1.

References

Butler, E.; Smith, A. M.; Liu, Y.-E.; and Popovic, Z. 2013. A mixed-initiative tool for designing level progressions in games. In *Symposium on User Interface and Software*.

Colton, S., and Wiggins, G. A. 2013. Computational creativity: The final frontier?

Cook, M., and Colton, S. 2013. From mechanics to meaning and back again: Exploring techniques for the contextualisation of code. In *Proceedings of the AIIDE Workshop on Artificial Intelligence and Game Aesthetics*.

Cook, M. 2013a. A rogue dream. In *Proceedings of the Fourth International Conference on Computational Creativity*.

Cook, M. 2013b. Spritely autogenerating sprites from the web. http://tinyurl.com/spritelypost.

Kopas, M. 2014. Lullaby for heartstick spacer. http://a-dire-fawn.itch.io/lullaby-for-heartsick-spacer.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2013. Sentient sketchbook: Computer-aided game level authoring. In *Proceedings of the ACM Conference on Foundations of Digital Games*.

Liu, H., and Singh, P. 2004. Conceptnet: A practical commonsense reasoning toolkit. *BT Technology Journal* 22:211–226.

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

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*.

Studios, B. G. 2011. The Elder Scrolls V: Skyrim. http://www.elderscrolls.com/skyrim.

Togelius, J.; Yannakakis, G. N.; Stanley, K. O.; and Browne, C. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions Computational Intelligence and AI in Games*.

Treanor, M.; Blackford, B.; Mateas, M.; and Bogost, I. 2012. Game-o-matic: Generating videogames that represent ideas. In *Proceedings of the Third Workshop on Procedural Content Generation in Games*.

Veale, T. 2012. From conceptual 'mash-ups' to 'bad-ass' blends: A robust computational model of conceptual blending. In *Proceedings of the 3rd International Conference on Computational Creativity*.