Chapter 12
Evaluating content generators (DRAFT)

Noor Shaker, Gillian Smith and Georgios N. Yannakakis

12.1 Motivation: I created a generator, now what?

The entirety of this book thus far has been focused on how to create procedural content generators, using a variety of techniques and for many different purposes. We hope that, by now, you've gained an appreciation for the strengths and weaknesses of different approaches to PCG, and also the surprises that can come from writing a generative system. We imagine that you also have experienced some of the frustration that can come from debugging a generative system: "is the interesting level I created a fluke, a result of a bug, or a genuine result?"

Creating a generator is one thing; evaluating it is another. Regardless of the method followed all generators shall be evaluated on their ability to achieve the desired goals of the designer (or the computational designer). This chapter reviews methods for achieving that. Arguably, the generation of any content is trivial; the generation of *valuable* content for the task at hand, on the other hand, is a rather challenging procedure.

What makes the evaluation of content (such as stories, levels, maps etc.) difficult is the subjective nature of players, their large diversity and, on the other end of the design process, the designer's variant intents, styles and goals [1]. Most importantly, content quality is affected by algorithmic stochasticity (such as metaheuristic search algorithms) and human stochasticity (such as unpredictable playing behavior, style and emotive responses) that affect content quality at large. All these factors are obviously hard to control in an empirical fashion.

In addition to factors that affect content quality there are hard (or softer) constraints put forward by the designers or the other game content elements that might be in conflict with the content generated (e.g. a proposed puzzle is not compatible with the level generated). A PCG algorithm needs to be able to satisfy designer constraints as part of its quality evaluation. Constrained satisfaction algorithms such as the feasible-infeasible two-population evolutionary algorithm used broadly by Liapis et al.[2] for mixed-initiative content creation and constrained solvers such as

answer set programming (see Chapter **??** of this book and [6]) are able to handle this.The generated results is within constraints, thereby valuable for the designer. *Value* however has variant degrees of success (depending on all the aforementioned factors) and this is where alternative methods or heuristics discussed in this chapter can help.

PGC can be viewed as a computational creator (either assisted or autonomous). One important aspect that has not been investigated in depth is the aesthetics and creativity of PCG within game design. How creative can an algorithm be? Is it deemed to have appreciation, skill, and imagination (Colton, 2008)? Evaluating creativity of current PCG algorithms a case can be made that most of them possess only skill. Does the creator manage to explore novel combinations within a constrained space thereby resulting in *exploratory* game design creativity (Boden); or, is on the other hand trying to break existing boundaries and constraints within game design to come up with entirely new designs, demonstrating *transformational* creativity (Boden)? If used in a mixed-initiative fashion, does it enhance the designer's creativity by boosting the possibility space for her? Arguably, the appropriateness of variant evaluation methods for autonomous PCG creation or mixed-initiative co-creation remains largely unexplored within both human and computational creativity research.

Content generators exhibit highly emergent behavior, making it difficult to understand what the results of a particular generation algorithm might be when designing the system. When making a PCG system, we are also creating a large amount of content for players to experience, thus it's important to be able to evaluate how successful the generator according to players who interact with the content.

The next section highlights a number of factors that make evaluating content generator important.

12.1.1 Why is evaluation important?

There are several main reasons that we want to be able to evaluate procedural content generation systems:

1. To better understand their capabilities. It is very hard to understand what the capabilities are of a content generator based solely on seeing individual instances of their output.
2. To confirm that we can make guarantees about generated content. If there are particular qualities of generated content that we want to be able to produce, it is important to be able to evaluate that those qualities are indeed present.
3. To more easily iterate upon the generator by seeing if what it is capable of creating matches the programmer's intent. As with any creative endeavor, creating a procedural content generator involves reflection, iteration, and evaluation.
4. To be able to compare content generators to each other, despite different approaches. As the community of people creating procedural content generators continues to grow, it's important to be able to understand how we are making progress in relationship to other people.

This chapter describes strategies for evaluating content generators, both in terms of their capabilities as generative systems and in performing evaluations of the content that they create. The most important concept to remember when thinking of how to evaluate a generator is the following: make sure that the method you use to evaluate your generator is relevant to what it is you want to investigate and evaluate. If you want to be able to make the claim that your generator produces a wide variety of content, choose a method that explicitly examines qualities of the generator rather than individual pieces of content. If you want to be able to make the claim that players of a game that incorporates your generator find the experience more engaging, then it is more appropriate to evaluate the generator using a method that includes the player.

12.2 Matching outcomes to design goals

One of the ultimate goal of evaluating content generators is to check their ability to meet the goals they are intended to achieve while being designed. Looking at individual samples gives a very high level overview of the capabilities of the generators but one would like for example to examine the frequency in which specific content is generated or the amount of variations in the designs produced by the system. It is therefore important to visualise the space of content covered by a generator. The effects of modifications made to the system can then be easily identified in the visualised content space as long as the dimensions according to which the content is plotted are carefully defined to reflect the goals intended when designing the system.

12.3 Expressivity measures

A tempting way to evaluate the quality of a content generator is to simply view the content it creates and evaluate the artifacts subjectively and informally. But if a content generator is capable of creating thousands, even millions, of unique levels, it is not feasible to view all of the output to judge whether or not the generator is performing as desired. If you see five levels that are impressive, among 50 that you choose to ignore or re-generate, what does that say about the qualities of the content generator?

To solve this problem, it is possible to evaluate the expressive range of the level generator. *Expressive range* refers to the space of potential levels that the generator is capable of creating, including how biased it is towards creating particular kinds of content in that space [7]. This evaluation is performed by choosing metrics along which the content can be evaluated, and using those metrics as axes to define the space of possible content. A large number of pieces of content are then generated and evaluated according to the defined metrics and plotted in a heatmap. This heat

map can reveal biases in the generator, and comparisons of the heatmap across different sets of input parameters can show how controllable the generator is.

- Being able to understand how controllable your generator is by seeing how expressive range shifts according to input changes.

12.3.1 Visualising expressive range

The expressive range of a content generator can be visualized as an N-dimensional space, where each dimension is a different quality of the generator that can be quantified. This allows us to imagine authoring level generators as creating these spaces of potential levels as a result of the emergent qualities of the system. By adding and removing rules from a rule-based generator, the shape of the generator's expressive range (also referred to as a generative space) can be altered.

For only two dimensions, the generative space can be visualized using a two-dimensional histogram. Higher dimensionality requires more sophisticated visualizations, which has not been deeply explored in the procedural content generation communities. This requires generating a representative sample of the content and ranking them according to the metrics; determining the amount of content to generate can be tricky. While it is simple for some systems to compute the total number of variations that can be generated, others may be able to create infinite variety. One method to ensure an acceptable sample size in the case of infinite content is to generate increasingly large amounts of content and visualizing expressive range, stopping when the graphs begin to look the same as the previous, smaller amount of content. Expressive range charts are not intended to be perfect, mathematical proofs of variety; rather, it is a visualization that can help the creator of a generative system to understand its behavior, and potential users of that system to understand its abilities.

Figure 12.1 shows the expressive range of the Launchpad level generator [8]. Notice that there is one large hot-spot for creating medium leniency, low linearity levels, and another bias towards creating medium leniency, high linearity levels (more on these metrics in the next section). Understanding that the system is biased towards these areas forces the designer of the system to ask why such biases exist.

Figure 12.2 presents another alternative method for visualising the expressive range of one of the content generators for Infinite Mario Bros [3]. The figure shows different distributions of the levels according to three expressive measures defined: linearity, leniency and density.

12.3.2 Choosing appropriate metrics

The metrics used for any content generator are bound to vary based on the domain that content is being generated for. The "linearity" and "leniency" metrics used in

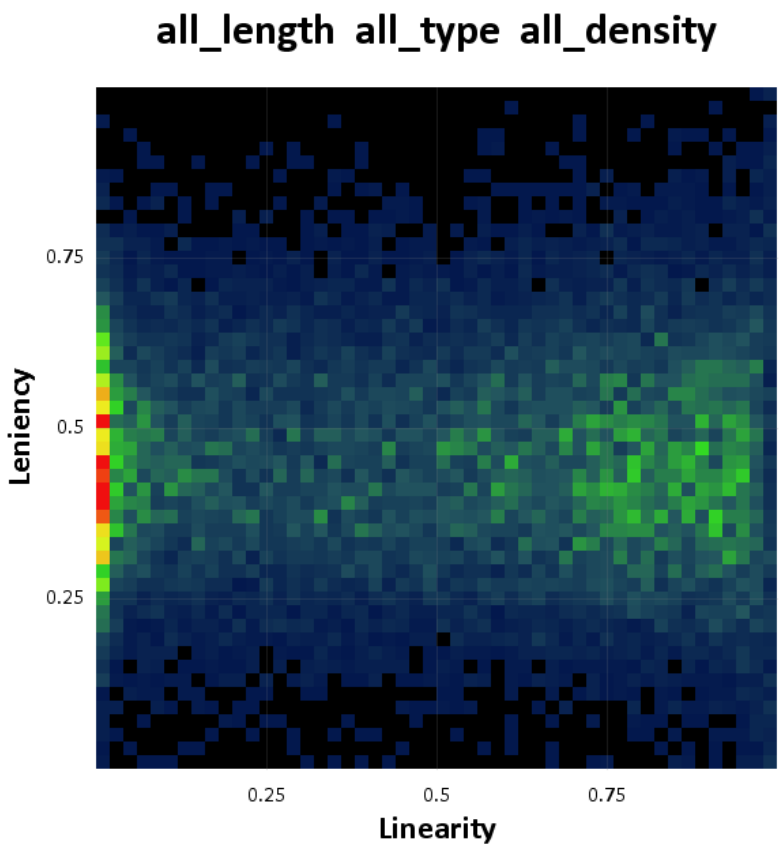

Fig. 12.1: The expressive range of the Launchpad level generator [8].

the Launchpad generator mentioned above make sense in the context of 2D platforming levels, but perhaps not in the context of weapons for a space shooting game.

The important rule of thumb to remember when choosing metrics for your content generator is this:

Strive to choose metrics that are as far as possible from the input parameters to the system. The goal of performing an expressive range evaluation is to understand the emergent properties of the generative system. Choosing a metric that is highly correlated to one that is used as an input parameter (e.g. if your generator accepts "difficulty" as an input and has "difficulty" as an expressive range metric) can only ever provide confirmatory results. If the system is specifically designed to create a particular kind of output, measuring for that output can only show that the algorithm operates as expected; it cannot deliver insight into unexpected behaviour or surprising output.

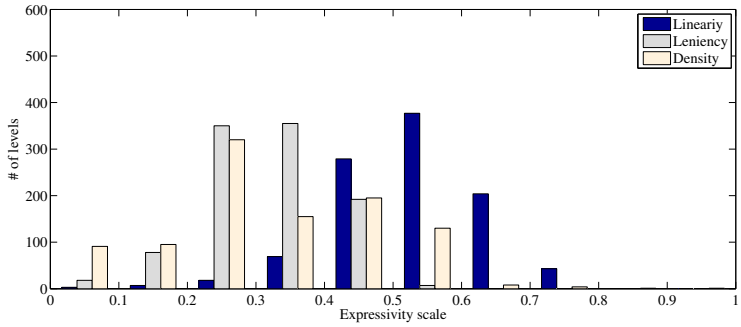

Fig. 12.2: The histograms of the linearity, leniency and density measures for one of the Infinite Mario Bros generator [3].

12.3.3 Understanding controllability

An important consideration in procedural content generation is understanding how well the generator can be controlled to produce different kinds of output, and especially how small changes in rule systems or priorities alters the expressivity of the system. If a designer requests that the system create shorter levels, will it still be capable of producing a broad range of content? Will adding a new rule to the grammar fundamentally alter the qualities of levels that can be produced?

Insight into these issues can be visualised by comparing expressive range graphs for different configurations of input parameters, either through visual comparison shows the expressive range of the Launchpad level generator when varying its rhythm input parameters (length of segment, pacing of segment, and type of rhythm) (see Figure 12.3). Notice that, while several graphs look quite similar to each other, there are notable parameter configurations that lead to drastically different resulting spaces. Gaining insight into the problem is helpful not only after creating a system and wanting to evaluate it, but also during the development process itself as a debugging tool.

12.4 User studies

Complementary to qualitative approaches to the evaluation of content generation, quantitative user studies can be of immense benefit for content quality assurance. The most obvious approach to evaluate the content experience by players (or designers) is to directly ask them about it. Within the subjective evaluation there are several schemes and questionnaires one can adopt. The general guidelines for self-report (or expert/designer-report) suggest that questionnaires should ask subjects to *rank* (and not rate) amongst various content experienced [9] as rating-based question-

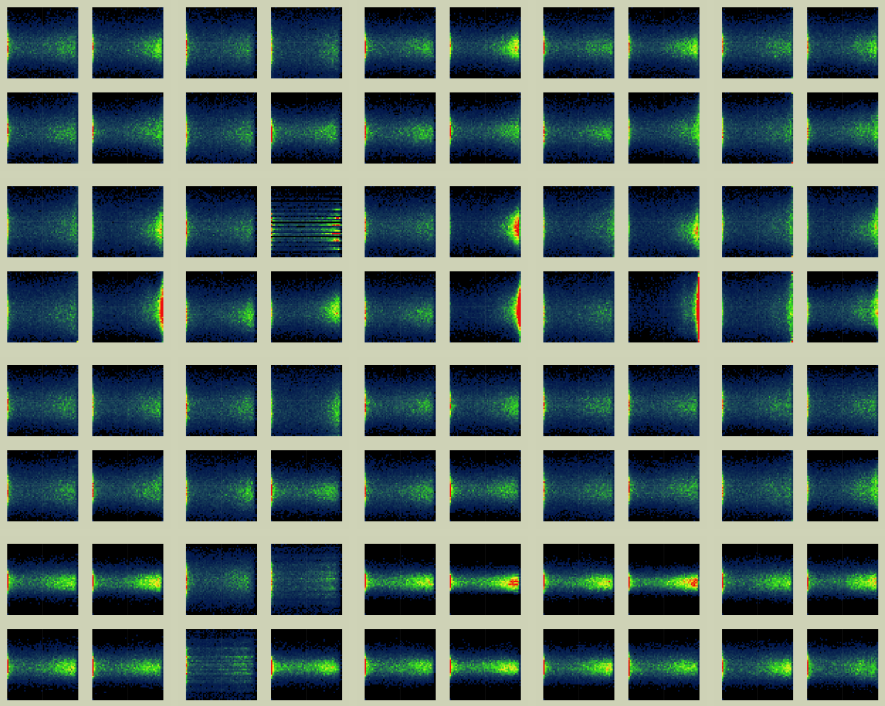

Fig. 12.3: The expressive range corresponding to different input parameters.

naires — such as the game experience questionnaire [**?**] — generate higher levels of inconsistency and order effects, lower inter-rater agreement, and are dominated by a number of critical biases that make any post analysis questionable (to say the least) [10, **?**]. The rank-based game survey approach has been successfully used in the Super Mario AI competition: level generation track [4]. The study can involve anything from a small number of dedicated players that will play through variant amounts of content to a crowd-sourced approach (see [5, **?**, **?**] among others. The numerous limitations of self-reporting can be eliminated via the use of rank-based questionnaires; however, not all issues of self-reporting can be surpassed this way. Self-reports can be replaced by or fused with alternate measures of player experience such as physiological manifestations (of e.g. arousal, interest and attention) and/or behavioral playing patterns that may map to particular a player state directly (e.g. a player which is stuck at the same map point for several rounds might be an indication of frustration). More details about objective measurement of player experience can be found in Chapter 9.

The data-driven (crowd-sourcing approach) has a number of limitations. The core objections against this method is the potential treatment of subjects as random content evaluators. Thus, ideally, the generator should be coupled with selection mechanisms that will prune the available content (which arguably can be generated in

massive amounts automatically) prior to it be presented to the players. On that basis, content can be evaluated (up to a good degree) via a sequence of logic operations without the need of player behavioral metrics or other input from players [**?**]. Further, the satisfaction of constraints or logical operators can be coupled with rapid simulations of AI agents that evaluate the quality of generated content (i.e.offline generate and test PCG).

12.5 Summary

References

1. Liapis, A., Yannakakis, G.N., Togelius, J.: Towards a generic method of evaluating game levels. In: Proceedings of the AAAI Artificial Intelligence for Interactive Digital Entertainment Conference (2013)
2. Loiacono, D., Cardamone, L., Lanzi, P.: Automatic track generation for high-end racing games using evolutionary computation. 3, pp. 245–259. IEEE (2011)
3. Shaker, N., Nicolau, M., Yannakakis, G.N., Togelius, J., ONeill, M.: Evolving levels for super mario bros using grammatical evolution. pp. 304–311 (2012)
4. Shaker, N., Togelius, J., Yannakakis, G.N., Weber, B., Shimizu, T., Hashiyama, T., Sorenson, N., Pasquier, P., Mawhorter, P., Takahashi, G., et al.: The 2010 mario ai championship: Level generation track. Computational Intelligence and AI in Games, IEEE Transactions on **3**(4), 332–347 (2011)
5. Shaker, N., Yannakakis, G., Togelius, J.: Crowd-sourcing the aesthetics of platform games. IEEE Transactions on Computational Intelligence and AI in Games (2013)
6. Smith, A.M., Mateas, M.: Answer set programming for procedural content generation: A design space approach. Computational Intelligence and AI in Games, IEEE Transactions on **3**(3), 187–200 (2011)
7. Smith, G., Whitehead, J.: Analyzing the expressive range of a level generator. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games, p. 4. ACM (2010)
8. Smith, G., Whitehead, J., Mateas, M., Treanor, M., March, J., Cha, M.: Launchpad: A rhythm-based level generator for 2-d platformers. Computational Intelligence and AI in Games, IEEE Transactions on **3**(1), 1–16 (2011)
9. Yannakakis, G.N.: Preference learning for affective modeling. In: International Conference on Affective Computing and Intelligent Interaction and Workshops, pp. 1–6. IEEE (2009)
10. Yannakakis, G.N., Hallam, J.: Ranking vs. preference: a comparative study of self-reporting. Affective Computing and Intelligent Interaction pp. 437–446 (2011)