Analyzing the Expressive Range of a Level Generator

Gillian Smith Jim Whitehead
Expressive Intelligence Studio
University of California, Santa Cruz
Santa Cruz, CA 95064
{gsmith, ejw}@soe.ucsc.edu

ABSTRACT

This paper explores a method for analyzing the expressive range of a procedural level generator, and applies this method to Launchpad, a level generator for 2D platformers. Instead of focusing on the number of levels that can be created or the amount of time it takes to create them, we instead examine the variety of generated levels and the impact of changing input parameters. With the rise in the popularity of PCG, it is important to be able to fairly evaluate and compare different generation techniques within similar domains. We have found that such analysis can also expose unexpected biases in the generation algorithm and holes in the expressive range that drive future work.

Categories and Subject Descriptors

I.2.1 **[Artificial Intelligence]**: Applications and Expert Systems – *Games*.

General Terms

Measurement, Experimentation.

Keywords

Procedural level generation, expressive range, evaluation methods.

1. INTRODUCTION

Procedural level generators typically excel at creating a large number of levels in a short period of time. However, it is impossible to judge the quality of a level generator based only on these statistics: a generator that can create tens of thousands of levels in a matter of minutes is useless if many of those levels are effectively identical to each other. It is instead better to judge the worth of a generator by the style and range of levels that it can create. The most common strategy for evaluating procedural content generators is to show examples of the kinds of content that can be produced. For example, showing different racetracks generated according to personalized fitness functions [5] or weapons that support different play styles [1]. This qualitative data may also be accompanied by statistics on the speed of the generator or the quantity of content that can be produced [2][4]. While this approach provides useful and interesting information about the generator, it does not fully capture the range of content that can be created and does not easily support analysis of how this range changes for different fitness functions or generation parameters. This paper presents a more rigorous approach to analyzing procedural content generators by classifying the style and variety of levels that can be generated: we call this quality of a generator its expressive range.

PCGames 2010, June 18, Monterey, CA, USA

We envision a future in which game designers, or even the players themselves, will want to choose different off-the-shelf level generators to include in their games. In this scenario, it is important to be able to satisfactorily describe each generator's expressive range, both to set expectations for the user and to make it easier to compare different generators. For example, one designer may wish to use a level generator that creates long, linear levels designed for speed runs, whereas another may want a generator that can create intricate levels for the player to explore.

Understanding the expressive range of a level generator is also useful in driving future work in level generation. It can uncover unexpected biases and dependencies in the generator, and show weaknesses in the variety of levels that can be produced.

We consider the following questions when judging the expressive range of a level generator:

1. What are appropriate ways to measure and compare produced levels?
2. How does the design of the generation algorithm itself affect the kinds of levels that can be produced?
3. What parameters should the designer expect to have control over, and how does altering these parameters impact the produced levels?
4. What are the extremes of the output of the system, and how can levels produced by it be compared?

This paper presents a framework for analyzing the expressive range of procedural content generators, and applies this approach to Launchpad, our level generator for 2D platformer games.

2. ANALYTICAL APPROACH

Our approach to analyzing the expressive range of a procedural level generator takes the following steps.

- ***Determine appropriate metrics.*** In order to evaluate the range of content that can be created, we must be able to measure and compare the results. To do this, we determine a set of metrics for describing levels. These metrics should be based on global properties of the levels, and ideally should be emergent qualities from the point of the view of the generator. This is because we would like to later view how the space changes by altering parameters to the generator.
- ***Generate content.*** Run the generator a large number of times to collect a representative sample of the generator's capabilities, scoring the content according to previously defined metrics.

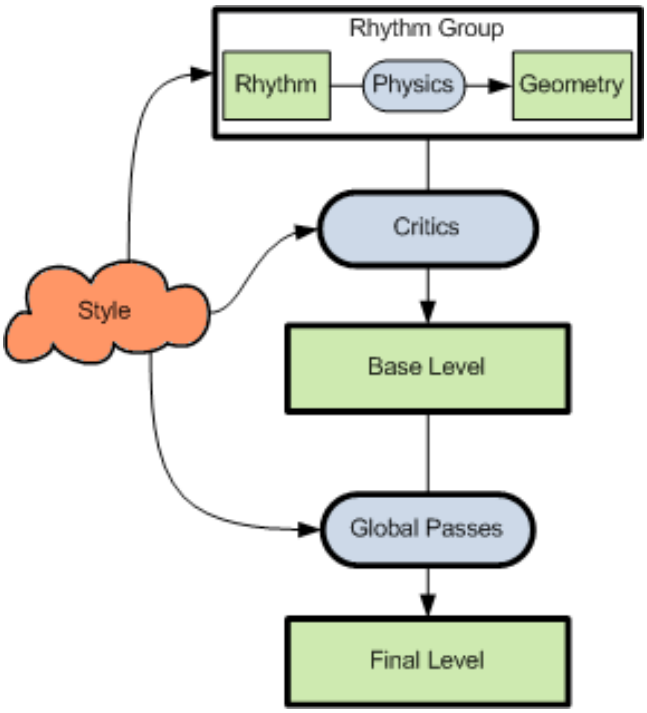

Figure 1. Level generation algorithm. Green squares indicate generated entities, while blue circles indicate constraints. Style parameters influence many aspects of level generation.

- ***Visualize generative space.*** The expressive range of the generator can be defined by the range of metrics scores. We suggest that one effective way to view this range is by creating a number of 2D histograms, where the axes are defined by the range of metrics scores. This allows us to view peaks of commonly created content and holes in the generative space.
- ***Analyze impact of parameters.*** We can now easily visualize how the expressive range of the generator is influenced by changing parameters and fitness functions applied to the generator, by comparing the difference between graphs of the generated space.

The remainder of this paper discusses how we apply this approach to Launchpad, an author-guided procedural level generator for 2D platformers.

3. SYSTEM OVERVIEW

Launchpad is a rhythm-based level generator for 2D platformer games [3], such as Super Mario World or Sonic the Hedgehog. The rhythm is that which players feel in their hands: for example, three short hops with minimal running in between, or three long jumps at a more leisurely pace. Levels are made up of a series of rhythm groups, which are short, non-overlapping sections of a level made up of a rhythm and geometry. Rhythm groups are constructed using a two-tiered, grammar-based approach. The first stage creates a set of player actions and assigns these actions to a specific time in a rhythm. The second stage uses a grammar to consume each action and replace it with geometry corresponding to that action, laid out according to a model of player movement, thus ensuring that all levels are playable. Launchpad creates a number of candidate levels which are then tested against designer-specified critics to determine the best level. A diagram describing Launchpad's generation algorithm is shown in Figure 1.

The designer has input to Launchpad in the form of manipulating "style" parameters: the line that the level should fit to, the frequency of certain geometry components being used, and the types of rhythms it can use to create levels. A table of these parameters is shown in Table 1.

3.1 Critics

To form a complete level, rhythm groups are fit together side by side, bridging them with a small platform that acts as a rest area for the player. Many different levels are generated, forming a pool of candidate levels which are then tested against a set of critics to determine the best level. These critics allow a designer to exert control over global properties of the level, and address a common problem with using design grammars: over-generation. Design grammars are good at capturing local constraints, but some design constraints are global. Given a generated level, a critic can perform tests over the entire level to determine how well a given global constraint is met. Our level generator has two critics: fitting to a designer-specified line (line distance), and ensuring the correct distribution of level components according to the designer's style

Table 1. Parameters for Launchpad that a user can manipulate.

Rhythm	*Type*	A rhythm can be "regular", "swing", or "random". Regular rhythms have evenly spaced beats, swing rhythms have a short followed by long beat, like a heartbeat. Random rhythms have randomly spaced beats.
	Density	Rhythm density describes how closely beats are spaced. Density can be "low", "medium", or "high".
	Length	Rhythms can be 5, 10, 15, or 20 seconds long.
	Action Probabilities	The probability of a jump action occurring vs. a wait.
Component	*Jump Geometry*	The probability of specific geometry being chosen for a jump action. Geometry available: jumping up or down, with or without gaps; enemy, spring, fall.
	Wait Geometry	The probability of specific geometry being chosen for a wait action. Geometry available: stomper, moving platform.
Line	*Equation*	The path that the user would like the final level to follow, defined as a set of non-overlapping line segments.
Critic	*Weighting*	The importance that Launchpad should place on the line distance critic vs. the component distance critic.

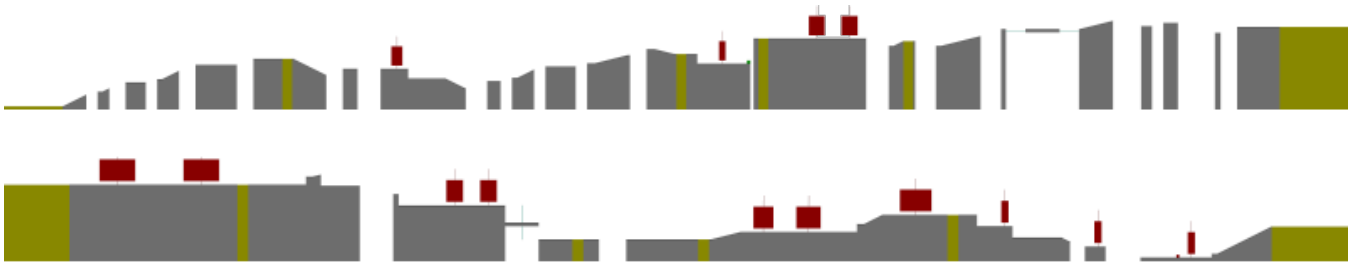

Figure 2. Two levels produced by Launchpad. Gray boxes are platforms, green boxes are platform joiners between rhythm groups Large red boxes are stompers, small red boxes are enemies, and small green boxes are springs.

specification (component distance). The importance of each critic can be adjusted to alter the kinds of levels produced.

Two examples of levels created by Launchpad are shown in Figure 2. Further details of the generator and a discussion of the role of critics can be found in our previous work [3]. This paper focuses instead on exploring Launchpad's expressivity and how designer control influences the space of levels it can create.

4. EXPRESSIVITY

This paper explores two ways to evaluate the expressive range of a level generator: by considering design implications inherent to the generation algorithm and by analyzing the space of levels that can be created. In this section, we present our approach by applying it to Launchpad, a system for creating 2D platformer levels.

4.1 Algorithm Implications

Certain kinds of platformer levels are excluded from Launchpad's expressive range due to our algorithm for level generation. In particular, we create levels that are dexterity-based rather than exploration-based. The challenge derived from these levels is more about perfectly timing movement through a series of obstacles, rather than seeking out hidden areas. Furthermore, we do not support the player choosing a path to take through the level, which is common in games likes Sonic the Hedgehog, and do not support the player turning around. Because of this, Launchpad's levels tend to favor a "speed run" play style.

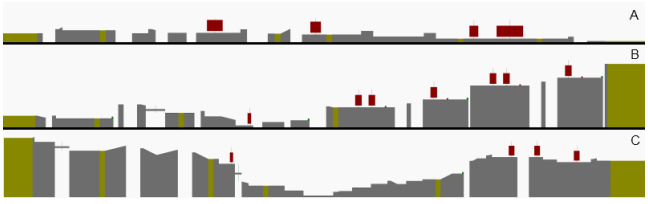

Figure 3. Three levels with different linearity scores. (A) Linearity = 0.05. (B) Linearity = 0.31 (C) Linearity = 0.67.

Figure 4. A level with that is highly linear (linearity = 0.1) but has a poor line distance critic value (line distance = 39.42). The pink line in the background is the control line that the level is supposed to follow.

4.2 Comparison Metrics

In order to describe the expressive range of a level generator, we must first be able to compare the levels that it produces. It is important that the metrics used for comparing levels are measuring emergent properties of levels, rather than simply using the same parameters that were used to guide the generator, so that we can see how input parameters impact the resulting levels. We define two different metrics for generated levels: linearity and leniency. We chose these metrics as they describe global qualities of levels, in terms of both their aesthetics and the gameplay. These qualities are useful for building an understanding of the generator's expressive range, but are not the only metrics that could be used. A similar analysis could be performed with more sophisticated measures such as the estimated time to completion or difficulty.

Linearity measures the "profile" of produced levels. We do this by fitting a line to the level and determining how well the geometry fits that line. The goal here is not to determine exactly what the line is, but rather to understand Launchpad's ability to produce levels that range between highly linear and highly non-linear. Examples of levels that fall at the extremes of this scale are shown in Figure 3. The linearity of a level is measured by performing linear regression, taking the center-points of each platform as a data point. We then score each level by taking the sum of the absolute values of the distance from each platform midpoint to its expected value on the line, and divide by the total number of points. Results are normalized to [0,1], where 0 is highly linear and 1 is highly non-linear. However, in our experiments, levels rarely had a linearity score higher than 0.7.

It is important to note that linearity and the line distance critic measure two different things, and that it is possible for a level to be judged "highly linear" for linearity but have a poor line distance score (Figure 4). The line distance critic is a measure for how well the level fits a designer-specified control line, whereas linearity measures the overall linearity of the produced level. Linearity is an aesthetic measure; line distance is a design heuristic.

Leniency describes how forgiving the level is likely to be to a player. We hesitate to quantify the difficulty of generated levels, as this is a subjective score and dependent on the specific ordering of component. However, it seems reasonable to describe levels that provide fewer ways for the player to come to harm as being more lenient than other levels. To measure this, we assign scores to each type of geometry that can be associated with a beat:

- +1.0: gaps, enemies, falls
- +0.5: springs, stompers

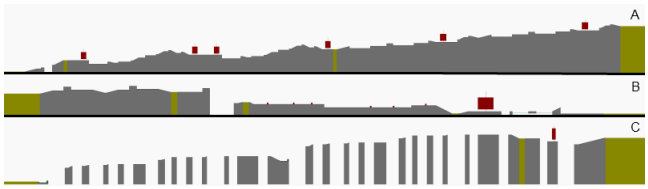

Figure 5. Three levels with different leniency scores. (A) Leniency = -0.75. (B) Leniency = 0.0 (C) Leniency = 0.95.

- -0.5: moving platforms
- -1.0: jumps with no gap associated

These scores are based on an intuitive sense of how "lenient" components are towards a player, with higher scores indicating less lenience. Example levels with different lenience scores are shown in Figure 5.

4.3 Expressive Range

With metrics allowing us to compare produced levels, we can describe the expressive range of the level generator by generating a number of levels and ranking them by their linearity and leniency scores. Figure 6 shows the expressive range for the generator when all components are weighted equally and all rhythms are being used. Each hexagon is colored to indicate the number of generated levels that have the corresponding linearity and leniency scores. All graphs used in this paper are based on 10,000 generated levels, unless stated otherwise.

The expressive range is clearly biased towards more linear levels, and slightly biased towards less lenient levels. The leniency bias is likely due to the greater number of non-lenient components. The linearity bias is a more interesting result, as we believe it is due to what was originally intended to be a small implementation detail in the level generator. When a component is chosen for inclusion in a rhythm group, the probability of that component appearing again is slightly increased. This detail was added late in the development of Launchpad, to fix a problem we perceived in our early levels: they did not have any discernable patterns, as we tend to see in games like Super Mario World where there tends to be locally repeated geometry. However, this approach means components that incur a height difference (e.g. jumping up to a new platform) are likely to stack up to create linear segments of the level. The linearity bias is an unintended side effect of this design decision. This issue highlights the importance of performing analysis as presented in this paper: we never would have realized such a minor change had such far-reaching effects without it.

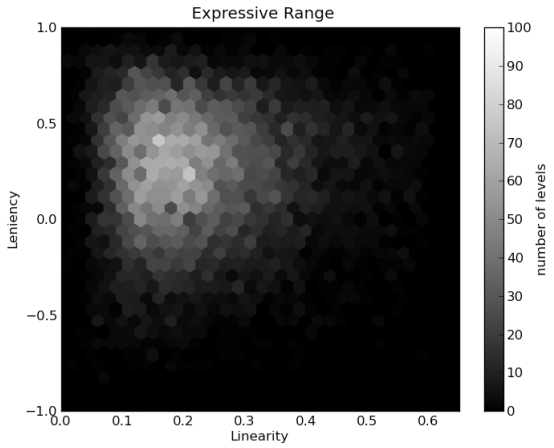

Figure 6. Launchpad's expressive range when all geometry types are weighted equally. Linearity is measured on the x axis, from 0.0 to 0.65. Leniency is measured on the y-axis, from -1.0 to 1.0. The color of each hexagon corresponds to the number of levels that have the associated linearity and leniency scores. The lighter the color, the more levels there are in that bin.

4.4 Influence of Generation Parameters

As noted in Table 1, there are a number of different parameters that can be varied to change the kinds of levels that Launchpad produces. In this section, we examine the impact on linearity and leniency when changing the rhythm and component parameters. Critics are not taken into account in this section: we look at all levels that are created, rather than only ones that surpass the critic threshold.

Rhythm type. We begin by varying the different kinds of rhythms used to generate levels and holding component probabilities constant and evenly weighted. Figure 7 shows the results of varying the rhythm type. The regular rhythm type offers the most variation of all the rhythm types, with no sharp peak in the graph. This distribution is what we had initially expected to see for all rhythm; however, it seems that swing and random rhythms are more constraining, contributing heavily to the bias in all rhythms. We hypothesize that these constraints are due to the potentially shorter amounts of time given to jumps in swing and random rhythms. This leads to fewer available actions, as a jump onto a spring requires much more time in the air than a jump across a short gap. For swing rhythms, this means that there will be fewer falls, springs, and moving platforms, contributing to a higher average leniency score.

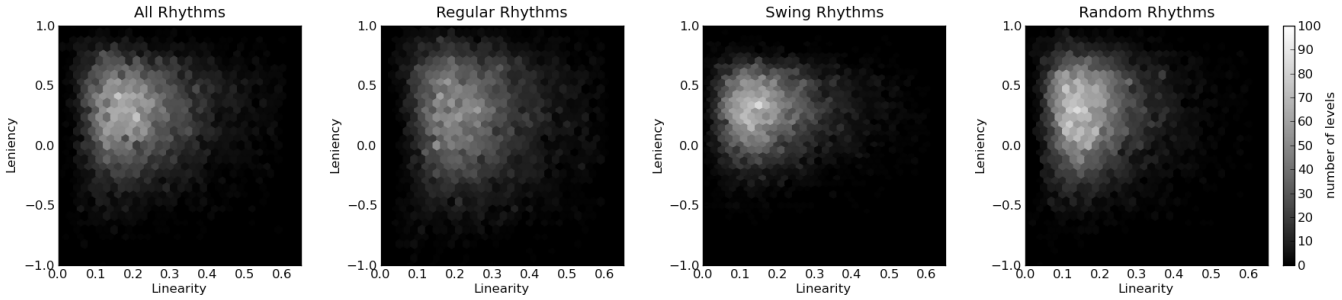

Figure 7. Varying rhythm type: (A) All rhythms, (B) only regular rhythms, (C) only swing rhythms, (D) only random rhythms.

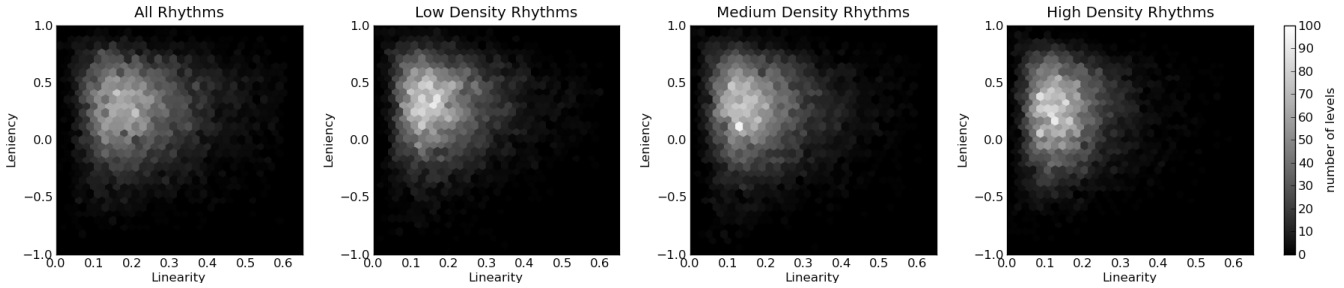

Figure 8. Varying rhythm density: (A) All rhythms, (B) low density, (C) medium density, (D) high density rhythms.

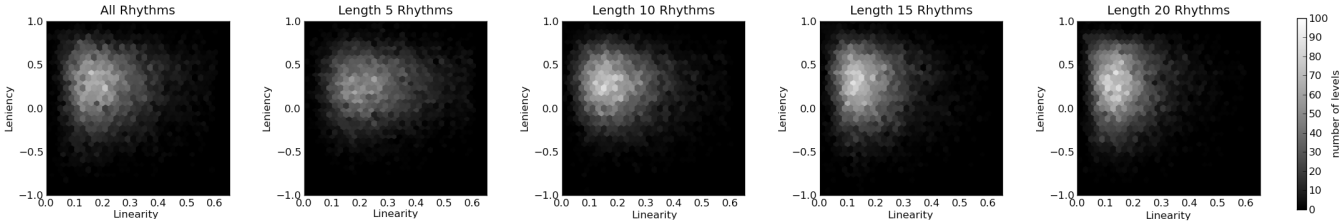

Figure 9. Varying rhythm length: (A) All rhythms, (B) 5 second, (C) 10 second, (D) 15 second, (E) 20 second rhythms.

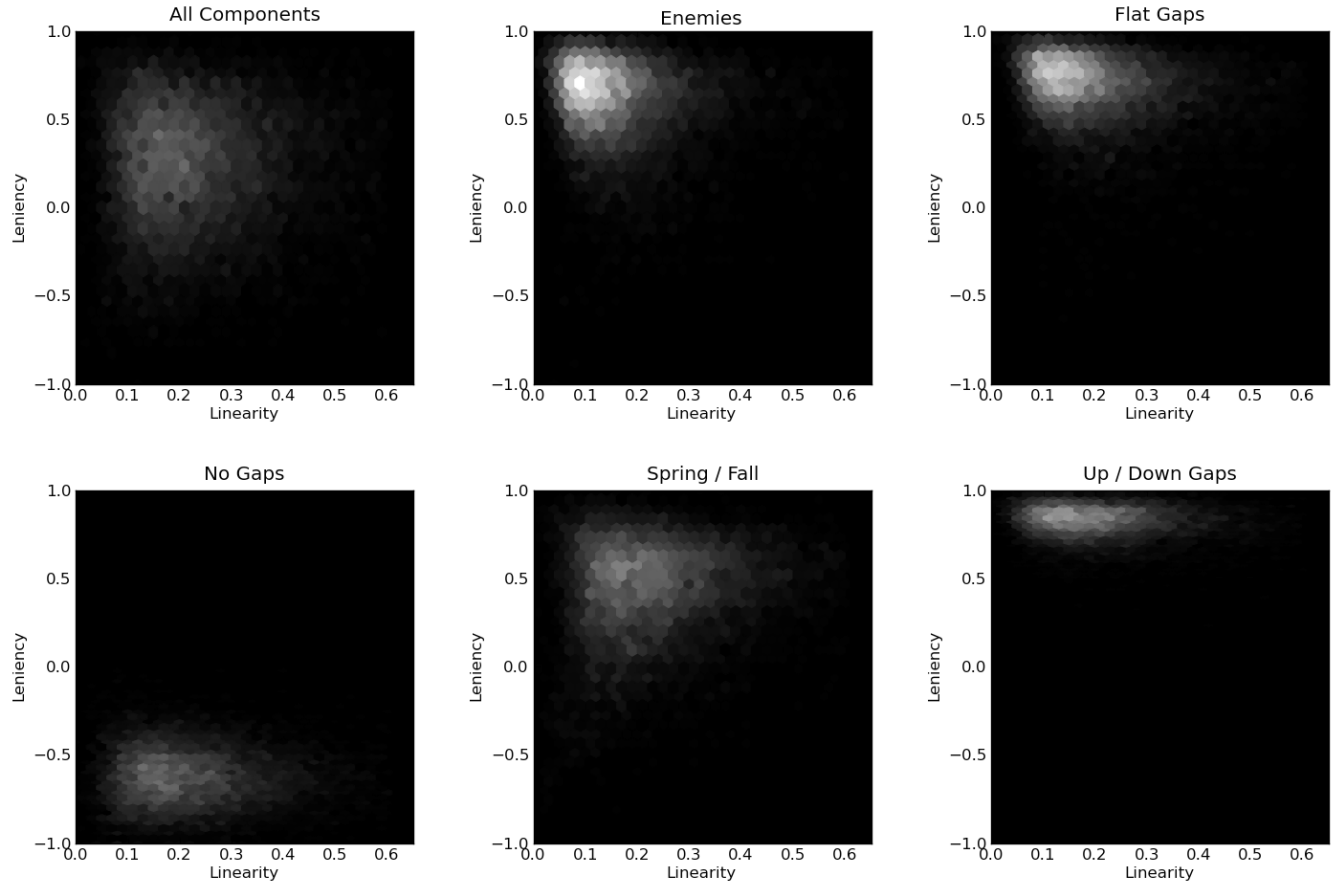

Figure 10. Varying chosen components. From top left to bottom right: (A) all components are weighted equally, (B) enemies are favored, (C) flat gaps are favored, (D) no gaps, (E) spring and fall are favored, (F) up and down jumps over gaps are favored. Colors scale from 0 levels to 200 levels.

Rhythm density. Figure 8 shows the results of varying rhythm density. Varying this parameter does not have a noticeable impact on leniency, but higher densities do lead to more linear levels. This is for the same reason that the generator is biased towards creating linear levels; higher density rhythms have more actions in them, and each action has a higher probability of the same component being chosen as before.

Rhythm length. Figure 9 shows the results of varying rhythm length. Again, there is a heavier bias towards creating more linear levels in longer rhythms, as longer rhythms tend to have more actions in them. Longer rhythms also seem to produce slightly more lenient levels; we hypothesize that this occurs for the same reason as the linearity bias: if the generator chooses a highly lenient component early in the rhythm group, the overall lenience of a rhythm group is likely to be higher. Since levels composed entirely of long rhythms have fewer total rhythm groups (since all levels that get generated are approximately the same length), this biases the generator towards creating more lenient levels.

Geometry Types. By varying the geometry types, we can see a drastic shift in the leniency of levels and a slight shift in linearity. Figure 10 shows the results of favoring a specific kind of component on the expressive range; other level components are allowed to be chosen, but with a very low probability. Leniency differences are easy to account for, as leniency is strongly correlated to the chosen components. There are also some interesting shifts in linearity of levels. Enemies and flat gaps tend to have mostly linear levels, due to the profiles of these components: there is little height difference at all. Similarly, gaps with a height difference tend towards have more non-linear levels, especially springs and falls, which have the largest height difference of all. The expressive range for favoring springs and falls also has the largest leniency difference: this is because springs and falls both take a large amount of time in the air, and so can be chosen less often as fewer rhythms support them. This means that other components appear more frequently in these levels than in other levels favoring different components.

4.5 Incorporating Critics

Until now, we have ignored the role of critics in analyzing Launchpad's expressive range. However, critics play an important role in allowing a designer to refine the space of levels that can be created. In this section, we examine the role of critics in evaluating expressivity.

Figure 11 shows the effect of filtering the linearity vs. leniency graphs we have been using by the line distance and component distance critics. Of the 10,000 levels that are generated, approximately 100 have a good critic measure: for line distance, this is a value less than 5, and for component distance, a value less than 40. Unsurprisingly, when filtering for levels that fit a straight line, the best levels are highly linear. However, it is important to note that filtering for component distance maintains a good range of both linearity and leniency. There is no noticeable difference in these graphs when changing the component frequency or control line parameters.

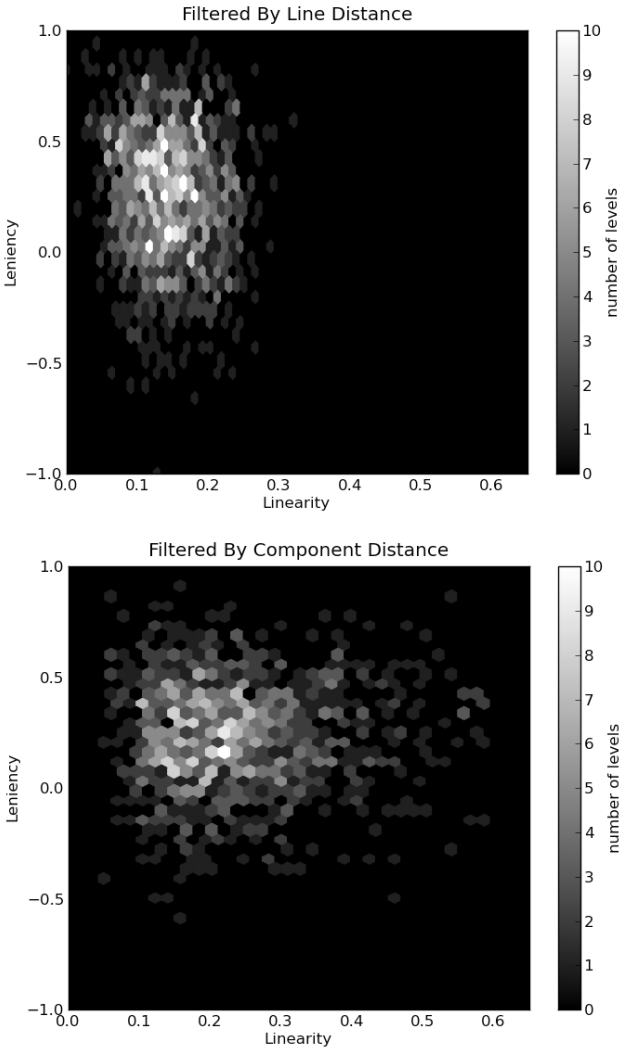

Figure 11. The effect on linearity vs. leniency when filtering for critic values. Colors describe the number of levels, where black is 0 levels and white is 10 levels. Top: only using levels that have a line distance score of less than 5. Bottom: only using levels that have a component distance score of less than 40.

5. DISCUSSION AND FUTURE WORK

Expressivity is a complex concept, especially with regards to computer-created content. We argue that expressivity emerges from both the generation algorithms and the input to a system, and that such expressivity can be unintentional. Humans assign meaning to the content that the system creates, without the system itself needing to understand this meaning. In other words, it is possible for a system to be *expressive* without necessarily being *creative*. For example, we do not argue that Launchpad is a *creative* system: although it can generate many unique levels, there is no meaning behind these levels beyond what we initially provide, i.e. rhythm. Launchpad is not capable of evaluating the quality of its output by any method other than pre-defined fitness functions. However, as we have shown in this paper, Launchpad is capable of creating a wide variety of levels, according to human-defined metrics, from relatively small building blocks.

As preliminary work in this area, there is clearly a great deal of potential future work. The first area is in determining better metrics for comparing levels. This is especially important when considering that a motivating factor for this work is being able to quantitatively compare two different level generators. While we feel that linearity

and leniency are important metrics for comparing platformer levels, they are not sufficient on their own and there is plenty of room for new and improved metrics. Solely aesthetic measures do not seem sufficient; a model of player behavior, perhaps encompassing different play styles, would be an interesting way of attacking this problem. Difficulty is an obvious measure, but other measures may include the perceived challenge due to pacing variation across the level, or different camera positions.

We have also focused exclusively on a search-based, offline level generator with minimal designer input. In online systems, it is less clear what should be compared for expressive range, as content changes over time in response to player actions. And in the presence of more sophisticated designer input, it becomes both more important to measure generator expressivity and less clear how it should be done. To accommodate a wide range of designers, the generator should be able to generate a wide range of content that fits a particular designer's style.

It is our hope that this paper will spur discussion on appropriate techniques for evaluating level generators, not just for the quality of individual levels, but for the entire range of levels that they can create.

6. ACKNOWLEDGMENTS

Our thanks to JD Stockford for his assistance in gathering data.

REFERENCES

[1] Hastings, E., Guha, R., and Stanley, K.O. 2009. Evolving Content in the Galactic Arms Race Video Game. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG '09)*. Milano, Italy. September 7 – 10, 2009.

[2] Hullett, K. and Mateas, M. 2009. Scenario Generation for Emergency Rescue Training Games. In *Proceedings of the 4th International Conference on Foundations of Digital Games*. Orlando, FL. April 26 – 30, 2009.

[3] Smith, G., Treanor, M., Whitehead, J., and Mateas, M. 2009. Rhythm-based Level Generation for 2D Platformers. In *Proceedings of the 4th International Conference on Foundations of Digital Games*. Orlando, FL. April 26 – 30, 2009.

[4] Tutenel, T., Smelik, R., Bidarra, R., and Jan De Kraker, K. 2009. Using Semantics to Improve the Design of Game Worlds. In *Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE '09)*. Stanford, CA. October 14 – 16, 2009.

[5] Togelius, J., De Nardi, R., and Lucas, S. 2007. Towards Automatic Personalised Content Creation for Racing Games. In *Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG '09)*. Honolulu, HI. 2007.