Combinatorial and Exploratory Creativity in Procedural Content Generation

Joris Dormans
Amsterdam University of Applied Sciences
Duivendrechtsekade 36-38
Amsterdam, The Netherlands
j.dormans@hva.nl

Stefan Leijnen
Amsterdam University of Applied Sciences
Duivendrechtsekade 36-38
Amsterdam, The Netherlands
s.leijnen@hva.nl

ABSTRACT

Procedural content generation aims to algorithmically produce creative solutions to game design challenges. This paper investigates how computational creativity theory can be applied to improve current PCG tools and techniques. It suggests that content generation may be considered as a dual process: a generation step to create variety and a resolution step to transform the output of the generation into a coherent and useful configuration. Separating these two steps facilitates the design of PCG algorithms and impacts the design of PCG tools.

1. INTRODUCTION

Procedural content generation (PCG) for games is a fast evolving discipline. Academic research constantly tries to push the boundaries of the field by improving techniques based on cellular automata [6], transformational grammars [3], evolutionary algorithms [13], or answer set programming [10]. Researchers also have explored the way procedural content generation can be used to create smart, "mixed-initiative" design tools to boost developer productivity [11, 9], automate the exploration of design space [8] or generate rhetorical mechanics [14], to name just a few examples.

In most (if not all) applications of procedural content generation the computer is responsible for finding solutions to design problems posed by the game's context, player's action, or designer's direction. In general, algorithms that can come up with the most 'creative' solutions will be considered best. In this case, we understand creative solutions to be fast and make sense within the context of the game, while they are still able to surprise both players and designers. This paper explores what it means for an algorithm to be creative, and how a more thorough understanding of artificial creativity can guide the design of new procedural content generation algorithms and tools. The paper sketches a new approach to procedural content generation that splits the creative process into two separate steps. Two simple experiments illustrate how this approach can be applied to procedural content generation. In the final section, the paper discusses how this approach can be used to improve automated game design tools that use procedural content generation techniques.

2. DESIGNING FOR CREATIVITY

Creative computational systems are often modeled after evolutionary processes, in which predefined sets of parameters are optimized in accordance with a selection function [5]. When these systems are used in PCG design tools, the choice of parameters and selection function becomes an optimization problem of its own: the designer of creative systems is required to balance out several features, such as

1. The size and structure of the solution space
2. The method for generating new solutions
3. The amount of time available for finding a sufficient solution
4. The criteria for deeming a solution sufficient

This paper deals with the transition from combinatorial to exploratory creativity: how can game designers develop procedural generation tools that go beyond merely permutating a set variables, while maintaining scalability and tractability? In computational creativity research, a distinction is often made between *combinatorial* creativity (combination of two previously unconnected elements) and *exploratory* creativity (exploration of an established conceptual space or style)[2, 7]

A second useful definition is to discern between *novelty* and *usefulness* [1]. These terms are reflected in the optimization design problem outlined above (i.e. how are new solutions generated? (novelty) What constitutes a solution? (usefulness)). This suggests that the generation of novelty and the generation of usefulness might be split into two different algorithms, and that the combination of those algorithms alongside the human design process allow for optimization of design at a higher level. At this level, the particular combination of quantitative factors (e.g. algorithm execution time, computational memory, human design time) and qualitative factors (i.e. novelty, usefulness) determine whether the creative process is combinatorial or exploratory. In the approach taken here, a simple algorithm (the generation step) is responsible for generating a wide variety of

data using a simple combinatorial logic. A second algorithm (the resolution step) is responsible for reorganizing the data into usable content for the game. Together, these two steps cooperate to push the creativity of the algorithm towards exploratory creativity.

Dividing the process into two steps with a clear division of responsibilities between them, has four important advantages:

1. The generation step becomes trivial to design and implement. In ideal situations any random combination will do, and in practice only a few simple constraints need to be taken into account.

2. The novelty of the produced content can be determined early on and before the computationally more expensive resolution algorithm is executed.

3. The designer of the generation algorithm only has to focus on specifying the structures that are allowed by the game. The designer does not have to specify all the possible combinations of structures, or how these combinations are to be generated.

4. The generation algorithm can be easily replaced by a different source of novelty. For example, it can be handcrafted by a designer, or produced in response to player performance.

3. EXPERIMENTS

Previous research into procedural content generation has led to the development of Ludoscope. Ludoscope is an experimental program that uses transformational grammars to generate content. It allows designers to create different types of grammars (string grammars, graph grammars, tile grammars, and shape grammars), and set up different recipes to generate content tailored towards a particular game. An important design feature of the program is its ability to split procedural content generation into multiple steps. Each step can be specified by different grammars, and produces different models that represent a game's content during various stages of its generation. Ludoscope's multi-step grammar-based operations and its relation to model driven engineering as general approach to procedural content generation has been reported previously [3, 4].

Initially, no changes to Ludoscope were needed to set up a number of experiments to test our assumptions about simple generation of novelty, combined with a more sophisticated resolution algorithm to generate a useful data set. The first experiment used a simple tile based grammar to generate dungeons that might be used in a roguelike game. The generation step simply produced a tile map randomly filled with walls and open spaces, except on the borders, which would always be walls (see figure 1). For this experiment, the chance that a tile (other than those on the edges) was set to a wall was 40%. The resolution step applies a simple transformation grammar to structure the random set into something that is more usable for a game (see figure 2). It is interesting to note that the results of the resolution slightly differ as the rules in the transformation grammar are applied randomly by selecting one possible transformation from all

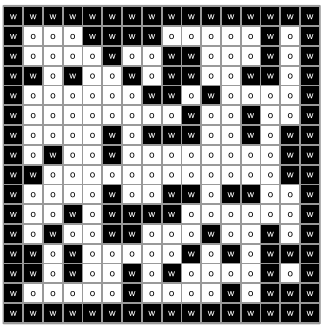

Figure 1: Randomly generated set of walls and open spaces to create a dungeon.

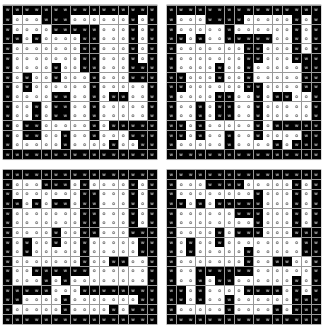

Figure 2: Sample results after the application an exploratory grammar to create more structure.

possible transformations every step. However, they do not differ as much as one might expect from a random application of transformation rules: the grammar converges on a number of stable solutions.

In many ways, the grammar for the resolution step acts as a cellular automaton: locally defined rules, depending on a tile's neighbors, change open spaces to walls and vice versa. However, it is important to note that the grammar used in the resolution step does not change the number of walls and open spaces as cellular automata would: open spaces and walls might swap places, but their respective numbers do not change. This feature is important when the same techniques are used to generate other elements in the dungeon at the same time. For example, when the number of monsters, doors and traps is determined during the same generation step.

The second experiment involves the generation of lock and key mission structures. In this case the generation step creates a string of tasks. The string always starts with an "entrance" and ends with a "goal" to represent the start end end points for the level. In addition, the first task is always a key and the last task is always a lock. This guarantees that any lock is preceded by at least one key and any key is followed by at least one lock. The intermediate tasks are randomly set to contain locks, keys or other task with probabilities of 25%, 25% and 50% (see figure 3). Next, the resolution step generates a structure in which each key is associated to at least one lock (and vice versa), and in which the initial sequence of locks, keys and other tasks is preserved. Figure

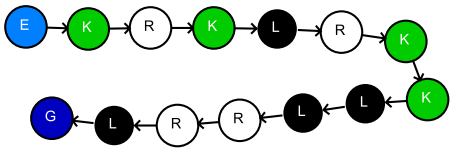

Figure 3: Randomly generated mission containing locks and keys (E = entrance, G = goal, L = lock, K = key, R = empty room).

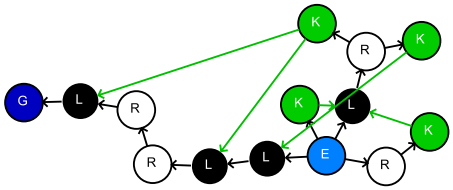

Figure 4: Sample result after the application an resolution grammar to create a spatial structure in which the mission of figure 3 is likely traversal. In this diagram green arrows indicate which keys unlock which locks, multiple keys might be required to unlock a single lock.

4 represents the outcome of this step. As with the previous experiment, the grammar for the resolution step does not change the number or types of nodes. It only changes their connections.

An important difference between this experiment and the previous experiment is that in this case the resolution step really does produces even fewer solutions for a particular generated set of locks and keys. This is guaranteed by executing each individual rule in the resolution grammar until they can no longer be applied to the the mission structure. In most cases the generated structure is always the same: for each input there is one result.[1]

A notable outcome of the second experiment is the ease with which the grammars generate a wide variety of different mission structures. The resolution grammar does not dictate how many keys are required to open a single lock, or how often it can be used to open multiple locks. Previous experiments with lock and key generation grammars required much more sophisticated grammars [3], yet did not generate the same variety of possible lock and key structures.

4. TAILORING PCG TOOLS

One important result of the new approach to procedural content generation sketched in this paper is a redesign of Ludoscope. Ludoscope originally was first and foremost a tool that helps designers create transformation grammars and experiment with different ways of execute multiple transformations. It was mostly relevant as a tool to design procedural content generation procedures for games. By explicitly supporting generation and resolution steps into this process, Ludoscope changed into a tool that fits in with "mixed-initiative" design tools [11], making it more relevant for game

[1]The grammar of this experiment only generates different results if its input contains multiple keys followed by multiple locks.

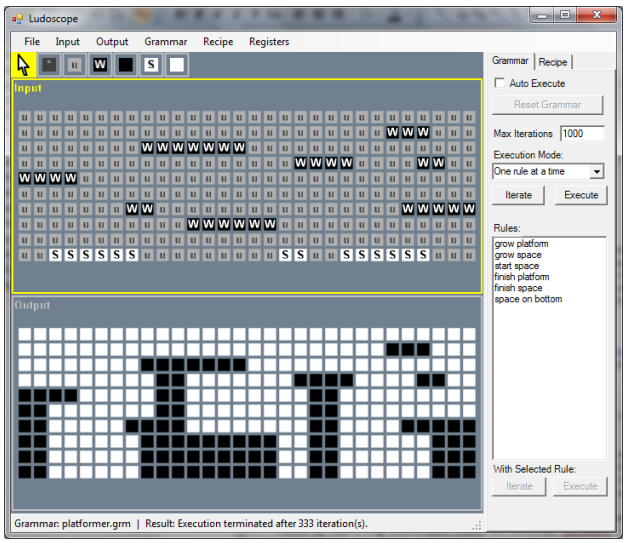

Figure 5: Latest version of Ludoscope featuring input and output channels.

development as a generic content production tool.

The most prominent change is the creation of input and output channels in the tool's main window (see figure 5). A designer is able to modify the model in the input channel (in the top half). Ludoscope uses that input to create an output. The output can be the result of the execution of a single transformation grammar, or a more complex procedure specified by a recipe that is able to execute multiple grammars and apply other special operations such as converting a graph to a figure consisting of two-dimensional shapes. Depending on the speed of the transformation, the output can be generated in response to any change in the input in real time.

For example, the output generated in figure 5 might represent a simple level for a platform game. In this case the designer specified the location of platforms in the input channel, and the transformation grammar fills in the details to create the complete level. In this case the designer executes the generation step manually and Ludoscope responds by filling the details; it executes the resolution step. This set-up allows a designer to explore different possibilities for distribution of platforms much faster than would normally be possible. In additions, grammars can be designed in such a way that it takes into account the distance a player might be able to jump in order to make sure the level remains playable. At the moment of writing we have only started to explore the possibilities this new approach brings.

Distinguishing between generation and resolution steps during content generation also makes it easier to change the source of variety for the algorithm. The variety can be deliberately designed manually, as is the case in Ludoscope. In addition, as was illustrated with the experiments above, simple random selection of elements already works if the resolution step is powerful enough to deal with (almost) all possible combinations. An options that is not explored in

detail in this paper is to use player input to generate levels. This is the case in the *Infinite Mario* experiment [12]. In this game, the locations where players jump, pick-up coins, or defeat enemies, are recorded and used as an input to generate the next the next level. Where in the original experiment, many weird and arguably less useful levels were generated in response to player actions, by following the player input with a resolution step, a much more consistent and playable game might emerge.

Furthermore, generation and resolution steps can be embedded within Ludoscope's original design philosophy of chaining transformations to break down the content generation into multi-step, feed-forward process [4]. In this case the output of a transformation serves as the input of the next transformation. In contrast to the previous approach, the new approach suggest that each generation step is followed by a resolution step and each resolution step is followed by a new generation step. In practice, this could mean that the first two steps generate a simple dungeon (for example the steps that were used to generate the dungeons in the first experiment) and are followed by a new generation step (for example to add traps, monsters and treasure) which in turn is followed by a new resolution step to make sure that the randomly added content is useful (for example monsters move to guard treasure, traps move to narrow passages, and so on). It is even possible that some generation steps are executed automatically, while others are based on user input. In the case of a process where designers can make changes to multiple steps in the process, this creates difficulties with reapplying changes that are made later in the process over changes that where made later in time but earlier in the process. The problem is to allow designers to edit multiple models that represent different perspectives on the same artifact [15]. It currently remains one of the more pressing research questions for the further development of Ludoscope.

5. CONCLUSIONS

Applying findings from the field of computational creativity to procedural content generation has yielded a number of interesting results. Breaking down the content generation algorithms into generation and resolution steps facilitates the design of procedural content generation algorithms. Generation and resolution steps can be applied to a wide variety of procedural content generation techniques. In this paper the focus was on a number of different grammar based approaches. However, cellular automata and evolutionary algorithms might equally benefit from separate generation and resolution steps. Evolutionary algorithms might be applied to generate interesting inputs before they are forwarded to a resolution algorithm.

This approach offers opportunities to determine the novelty and usefulness of a generated solution early on, before the more computationally expensive resolution step is executed. This can speed up the generation procedure as a whole considerably. Ideally the resolution step is designed not to converge or diverge any further, but to stabilize the variety. This makes the resolution highly controllable and suggests new opportunities to create mixed initiative design tools. In this case, designers are responsible for creating new content and the tools automatically resolve the designer's input according the game's constraints.

6. REFERENCES

[1] M. Boden. Allen newell and j. g. shaw and herbert a. simon. In H. E. Gruber, G. Terrell, and M. Wertheimer, editors, *Contemporary Approaches to Creative Thinking*, pages 63–119. Atherton, New York, NY, 1963.

[2] M. Boden. *The Creative Mind: Myths and Mechanisms*. Routledge, New York, NY, 2004.

[3] J. Dormans. Adventures in level design: Generating missions and spaces for action adventure games. In *Proceedings of the Procedural Content Generation Workshop 2010, Monterey, CA*, 2010.

[4] J. Dormans. Level design as model transformation: A strategy for automated content generation. In *Proceedings of the Procedural Content Generation Workshop 2011, Bordeaux, France*, 2011.

[5] J. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, Cambridge, MA, 1992.

[6] L. Johnson, G. N. Yannakakis, and J. Togelius. Cellular automata for real-time generation of infinite cave levels. In *Proceedings of the Foundations of Digital Games Conference 2010, Monterey, CA*, 2010.

[7] D. Novitz. Creativity and constraint. *Australasian Journal of Philosophy*, 77:67–82, 1999.

[8] A. Pantaleev. In search of patterns: Disrupting rpg classes through procedural content generation. In *Proceedings of the Procedural Content Generation Workshop 2012, Raleigh, NC*, 2012.

[9] R. Smelik, T. Turenel, K. J. de Kraker, and R. Bidarra. Inegrating procedural generation and manual editing of virtual worlds. In *Proceedings of the Procedural Content Generation Workshop 2010, Monterey, CA*, 2010.

[10] A. M. Smith and M. Mateas. Answer set programming for procedural content generation: A design space approach. *Computational Intelligence and AI in Games, IEEE Transactions on*, 3(3):187 –200, sept. 2011.

[11] G. Smith, J. Whitehead, and M. Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Procedural Content Generation Workshop 2010, Monterey, CA*, pages 209–216, 2010.

[12] J. Togelius, E. Kastbjerg, D. Schedl, and G. N. Yannakakis. What is procedural content generation? mario on the borderline. In *Proceedings of the Procedural Content Generation Workshop 2011, Bordeaux, France*, 2011.

[13] J. Togelius, M. Preuss, and G. N. Yannakakis. Towards multiobjective procedural map generation. In *Proceedings of the Foundations of Digital Games Conference 2010, Monterey, CA*, 2010.

[14] M. Treanor, B. Blackford, M. Mateas, and I. Bogost. Game-o-matic: Generating videogames that represent ideas. In *Proceedings of the Foundations of Digital Games Conference 2012, Raleigh, NC*, 2012.

[15] R. Walter and K. Neuwald. Calliope-d: An authoring environment for interactive dialog. In H. Reiterer and O. Deussen, editors, *Mensch & Computer 2012: interaktiv informiert - allgegenwärtig und allumfassend!?*, pages 409–418, MÃijnchen, 2012. Oldenbourg Verlag.