Procedural Content Generation Using Patterns as Objectives

Steve Dahlskog[1], Julian Togelius[2]

[1] Malmö University, Ö. Varvsgatan 11a, Malmö, Sweden
[2] IT University of Copenhagen, Rued Langaards Vej 7, 2300 Copenhagen, Denmark
steve.dahlskog@mah.se, julian@togelius.com

Abstract. In this paper we present a search-based approach for procedural generation of game levels that represents levels as sequences of *micro-patterns* and searched for *meso-patterns*. The micro-patterns are "slices" of original human-designed levels from an existing game, whereas the meso-patters are abstractions of common design patterns seen in the same levels. This method generates levels that are similar in style to the levels from which the original patterns were extracted, while still allowing for considerable variation in the geometry of the generated levels. The evolutionary method for generating the levels was tested extensively to investigate the distribution of micro-patterns used and meso-patterns found.

1 Introduction

The study of Procedural Content Generation (PCG), i.e. how game content such as levels, items, quests and characters can be created algorithmically, is currently one of the most active topics within academic research on artificial and computational intelligence in games. A large variety of methods have been proposed to generate an even larger variety of types of game content, subject to various objectives and constraints [1]. The work is motivated both by a real industry need for lowering the cost and saving time of content production and enabling endless user-adaptive games, and by academic interest in formalising game design and building creative machines. A recent "vision paper" for PCG research lists a number of open research challenges [2]. One of them is to learn to imitate style: could you build a content generator that was shown a number of examples of the creative output of a human or team of humans, and that then learned to produce more artefacts in the same style that were clearly original but still recognisably of the same style?

Another active research area has been that of game design patterns. A design pattern is a general concept, which has its roots in architecture, but has been applied both to software design and to game design. Game design patterns have so far been identified manually, and the investigation on how to integrate patterns into PCG has just started.

In this paper we demonstrate how practical game design patterns can be combined with procedural content generation to generate game levels that imitate a certain design style, and report the results of a series of experiments using a platform game benchmark. We have previously analysed the classic game *Super Mario Bros.* (SMB) [3] and suggested a collection of patterns and a PCG tool that produce levels by randomly picking copies of these patterns and modifying them according to a desired length and difficulty level [4].

Our prototype is based on evolutionary computation, where we will search the solution space of combinations of simple building blocks for levels that contain structures at a higher level. This way, we introduce a certain measure of control and constrain the shape of the final level through both the objective function and the choice of building blocks, while allowing a significant amount of variation. In the prototype the representation is relying on existing content in SMB, namely on one tile wide *vertical slices*, which we will also refer to as *micro-patterns*. The micro-patterns are extracted from the original SMB levels. A level is simply a sequence (or string) of micro-patterns — this applies both to the original levels and our generated levels. However, not any sequence is interesting but in our prototype we search for specific sequences or patterns that exists in the original game. These sequences will we refer to as *meso-patterns* and they are our search objective for our evolutionary approach.

We have previously reported initial work on this idea in a workshop paper [5]. Compared to that paper, the current paper describes a more mature system, and reports more in-depth results with several variations of the fitness function and a better characterisation of the generator output.

1.1 Background

In the seventies, Alexander et al. proposed a pattern language for architectural application on all levels (regions, cities, neighbourhoods, buildings and rooms) thus allowing everybody the ability to express design. Not only structural and material issues are covered but also life experience like the *Street Cafe*-pattern. The pattern language consists of a set of problems in an environment together with a core solution to its corresponding problem [6] thus giving a designer a tool to handle reoccurring problems. This powerful idea has spread to other areas like object-oriented software development where Gamma et al. have defined a set of templates for solving general design and programming problems [7]. In the context of games have Björk and Holopainen suggested an extensive collection of patterns for game design [8]. Similarly, others have looked into game mechanics [9] and specific game contexts like FPSs [10], RPGs [11], and action games [12]. There have also been some attempts to formulate abstract level design patterns that can be specialised to concrete metrics for different level types [13].

Procedural content generation refers to the (semi-)automatic process of creating game content. One common approach to PCG is the search-based approach, to use evolutionary computation or other stochastic global search/optimisation algorithms [14] for searching the content space. An oft-encountered trade-off in PCG is between control and variation. Methods that have a high variation in output according to some measure usually afford little designer control. Variation can be measured as *expressive range*, the variation along relevant metrics of generated artefacts [15, 16]. Control comes in several flavours: control over style, player experience, difficulty or even playability (e.g. specifying that there is a path from start to end of a level).

1.2 Examples of patterns

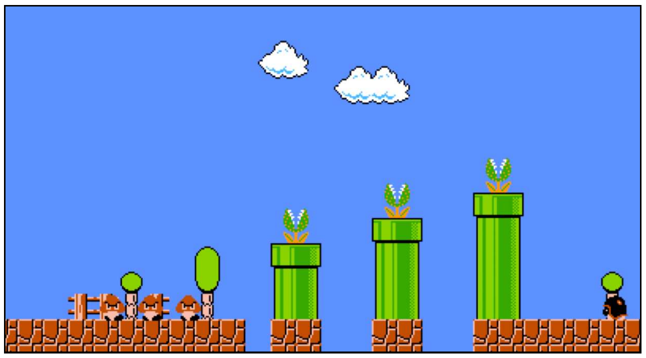

Fig. 1. Three consecutive patterns in SMB.

Because of the limited space available we can only briefly mention the patterns that were found [4] in (SMB). The patterns can be grouped into 5 groups; 1) Enemies and hordes, (single and multiple variations), 2) Gaps (single, multiple, variable length, combined with enemies and structures), 3) Valleys (a boxed-in area with structures, possible combined with enemies), 4) Multiple paths (structures horizontally dividing game space combined with enemies and rewards) and 5) Stairs (structures supporting vertical repositioning combined with enemies and gaps). In figure 2 we can see two instances of the 3-Horde pattern (Enemies) and in figure 1 we have a 3-Horde-pattern, a Pillar Gap-pattern and a Enemy-pattern.

2 Rationale

Our application domain in this paper is the classic 2-dimensional platformer, *Super Mario Bros.* (SMB) [3] and our generator is implemented using the Java-based Mario AI Benchmark[3] [17].

The levels of SMB could be seen as 2D matrices where the cells contain various items such as blocks, coins, enemies, etc.; this is also the internal representation of levels in the Mario AI benchmark. Mario (when small) has the size

[3] The benchmark is based on the clone *Infinite Mario Bros* by Markus "Notch" Persson.

of 1 cell, and most levels have a length of 100-300 cells and a height of 20 cells. A slice, or micro-pattern, is simply a vertical column of this array – a subarray with length 1. By analysing the levels of the original SMB, we have identified a library of such slices. New levels could be created by combining slices from this library, drawn at random. Such levels would have some similarity to the original levels, as they would not contain any slices that did not exist in the original game. They would not, for example, contain slices where enemies stack on top of each other or the player starts in mid-air. However, these levels would be uninteresting at best, and probably unplayable, as they might contain too long gaps, unclimbable walls, long stretches of nothing, and generally no discernible structure. However, in the space of all possible sequences of slices there should be many permutations that are well-designed, playable levels that are similar to the original SMB levels not only on micro level but also on meso- and macro-levels. How can we find those levels? In order to guarantee playability we punish unplayable sequences.

2.1 Representation

Our level representation is a sequence of symbols of length 200, where each symbol stands for a specific *micro-pattern* (a vertical slice) taken from the original human created content. The slice is one tile wide and in our example we have a slice containing a Goomba standing on a ground tile. This tile could be copied in sequence two or three times to make a 2-Horde or 3-Horde pattern (as in fig. 2).

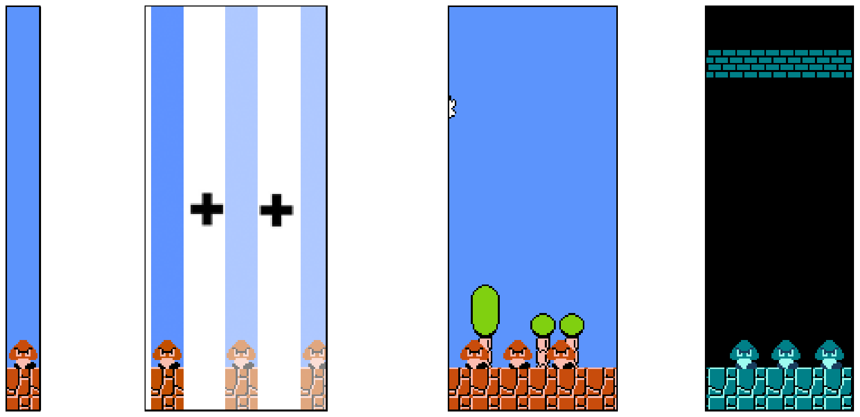

Fig. 2. To the far left we have a vertical slice (micro-pattern) with a Goomba on low ground. To the left a sequence of copies of the same slice making up a 3-Horde meso-pattern that in the original game can be found quite often as in World 8, Level 1 seen to the centre-right and in World 1, Level 2 to the far right.

By adding new slices the solution space grows. The levels of the original SMB contain fewer than 200 slices like this. In our representation, we use an alphabet consisting of 23 frequently occurring micro-patterns. Most of the slices come from unique-looking levels like W1L2 (the first level under ground) and are not reused elsewhere in the game. The advantage of the representation is the ease with which one can generate a level either by the constructive or the generate-and-test approach [14]. One could for example base a constructive PCG algorithm on a *phrase-structure grammar* with pre-checked production rules or by randomly picking slices and evaluate according to constraints. However, we will suggest another approach in the next section.

2.2 Evolutionary algorithm

The search-based approach taken in this paper is based on a fitness function that rewards the presence of meso-patterns, the higher presence the likelier a member is selected. We apply a simple $\mu + \lambda$ evolution strategy where $\mu = \lambda = 100$ is combined with single-point mutation and one-point crossover. In other words, of a population of 200 we apply *selection* (discarding half of the population), *reproduction* (keeping half of the population and using pairwise breeding to generate new members), *recombination* (fixed one-point-crossover) and *mutation* (the slice at a randomly chosen position in the level has its symbol replaced by a randomly chosen slice).

2.3 Fitness function

In order to understand how our micro- and meso-patterns interact in the search space we implemented three fitness functions (FF 1-3). The fitness functions were designed in the following way; FF1) a simple uniform reward value for every *unique pattern*, FF2) a simple uniform reward value for *every occurrence of patterns*, and finally FF3) a non-uniform reward weighted value for every occurrence of patterns. The first fitness function worked as a validation of the strings indicating that they could be found (i.e. more than one out of our meso-patterns can be found). The second fitness function was used to explore the frequency of how meso-patterns "appear" in the search space (i.e. how common are the different meso-patterns). The third fitness function was used to explore how the use of weighted values affects the frequency of meso-patterns.

In order to have some input on the weights to use we chose a simple strategy of calculate a weight by inverting the average occurrence of the patterns giving an infrequent pattern a high weight and a frequent pattern a low weight. By doing so, we propose that we can counter the effect of normal distribution while picking random symbols during the task of initiating and mutating the members of the population. Another issue this strategy would counter, is the varying complexity that the individual patterns have. If we would continue to use a uniform reward strategy for the fitness function, complex strings would run a greater risk to be starved to death in our population due the space it takes over uncomplicated patterns (i.e. short patterns are easily fitted into a member in relation to a long pattern). In order to find different variations of the patterns we designed a set of 43 strings of symbols in different categories of the patterns (i.e. 5 categories of patterns and 23 patterns [4]). These strings, (which we will refer to as rules) were used for a simple linear search, covering each member of the population in each generation.

3 Results and evaluation

We performed the experiments in three stages. First, we evolved a large number of levels using the “unique patterns” version of the evaluation function (FF1). We then repeated this experiment using the “all occurrences” version of the evaluation function (FF2). Based on these runs, we evaluated which micro-patterns were most commonly used, and which meso-patterns were most commonly found. These evaluations were used to calculate the weights for a weighted version of the fitness function (FF3). The third and final experiment, using the weighted version of the evaluation function, aimed to see if we could bring about that all patterns were found in a more balanced way.

3.1 Finding patterns

Table 1. Fitness value variation for 1000 levels counting fitness value based on rules; only one occurrence (FF1), multiple occurrences (FF2) and weighted multiple occurrences (FF3).

Generations	MIN	MAX	MEAN	DEV.	MED.
0 (FF1)	3	8	4.61	0.81	5
10 (FF1)	5	11	7.47	1.02	7
100 (FF1)	8	27	14.94	2.51	15
500 (FF1)	8	31	18.18	3.17	18
1000 (FF1)	9	31	18.97	3.23	19
0 (FF2)	4	10	5.7	1.12	6
10 (FF2)	7	18	11.17	1.74	11
100 (FF2)	13	86	36.98	10.46	37
500 (FF2)	16	183	68.62	30.17	63
1000 (FF2)	18	227	82.17	37.38	73
0 (FF3)	4	202	77.83	36.16	77
10 (FF3)	8	301	121.92	62.83	118
100 (FF3)	20	1030	264.07	149.64	241
500 (FF3)	34	2361	430.33	348.98	337
1000 (FF3)	34	2449	486.20	401.76	374

For each fitness function, we made 1000 independent runs and recorded the fitness values based on the strings. The fitness value worked as a simple ”count a rule when it is fulfilled”, but only the first time it occur in a level for FF1, for every time it occurred in FF2 and with weighted values in FF3. We can see that the evolutionary approach manages to find more meso-patterns over time. In order to measure the effect of our efforts of guiding the evolution to find more elaborate patterns we recorded which rules were present in the best member out of our 1000 runs (see table 2).

Measuring the occurrences of a rule in large population should give an indication on how complicated it is to generate an instance of a meso-pattern (rule) in relation to the micro-patterns. Several of the meso-patterns use the same micro-patterns and since the micro-patterns initial occurrence is based on equal chance to be present in the population and a member we can be certain that, given enough time, the search-based approach will affect the distribution of micro-patterns.

Table 2. Found patterns (rules) in FF1-FF3 together with the calculated weight for FF3 based on 1000 runs.

Pattern	Mesa		Straight	Multi-way											
Occurrence in FF1	682	686	1001	239	193	50	93	68	193	168	239	197	132	136	
Average in FF1	0.68	0.69	1.00	0.24	0.19	0.05	0.09	0.07	0.19	0.17	0.24	0.20	0.13	0.14	
Occurrence in FF2	498	480	523	25	83	221	329	11	83	37	25	13	120	127	
Average in FF2	0.5	0.48	0.52	0.03	0.08	0.22	0.33	0.01	0.08	0.04	0.03	0.01	0.12	0.13	
Weight	2.01	2.08	1.91	40	12.05	4.53	3.04	90.91	12.05	27.03	40	76.92	8.33	7.87	
Occurrence in FF3	1042	1118	1317	574	264	317	*40*	559	264	298	574	589	697	687	
Average in FF3	1.04	1.12	1.32	0.57	0.26	0.32	0.04	0.56	0.26	0.30	0.57	0.59	0.70	0.69	
Pattern	Enemy						Hordes					Gaps			
Occurrence in FF1	2605	1198	572	2606	1208	525	920	931	1007	1007	892	111	286	269	286
Average in FF1	2.61	1.20	0.57	2.61	1.21	0.53	0.92	0.93	1.01	1.01	0.89	0.11	0.29	0.27	0.29
Occurrence in FF2	13751	10411	1897	13584	8678	722	3694	4995	8209	8209	3563	14	83	68	132
Average in FF2	13.75	10.4	1.9	13.6	8.68	0.72	3.69	5	8.21	8.21	3.56	0.01	0.08	0.07	0.13
Weight	0.07	0.1	0.53	0.07	0.12	1.39	0.27	0.2	0.12	0.12	0.28	71.43	12.05	14.71	7.58
Occurrence in FF3	*444*	*50*	*8*	*444*	*33*	*16*	*0*	*90*	*93*	*93*	*0*	1720	*44*	*33*	*88*
Average in FF3	0.44	0.05	0.01	0.44	0.03	0.02	0.00	0.09	0.09	0.09	0.00	1.72	0.04	0.03	0.09
Pattern	Valley			Stair					Pipes						
Occurrence in FF1	87	81	61	845	846	664	705	716	66	47	43	46	61	67	
Average in FF1	0.09	0.08	0.06	0.85	0.85	0.66	0.71	0.72	0.07	0.05	0.04	0.05	0.06	0.07	
Occurrence in FF2	17	14	17	355	352	257	289	287	28	14	9	14	8	10	
Average in FF2	0.02	0.01	0.02	0.36	0.35	0.26	0.29	0.29	0.03	0.01	0.01	0.01	0.01	0.01	
Weight	58.82	71.43	58.82	2.82	2.84	3.89	3.46	3.48	35.71	71.43	111.1	71.43	125	100	
Occurence in FF3	193	178	162	1233	1197	1110	915	1025	*5*	43	57	**12**	966	30	
Average in FF3	0.19	0.18	0.16	1.23	1.20	1.11	0.92	1.03	0.01	0.04	0.06	0.01	0.97	0.03	

For FF1, the distribution of fulfilled rules show promise on only 12 of the rules (with occurrence value of 845–2605) and all rules have been fulfilled. However, this is not sufficient to answer the question on how easy they are to find in relation to each other. It is possible that the more complex rules are starved to death in an evolutionary search. In order to explore this we ran FF2 and counted multiple occurrences. The effect of counting multiple instances gives the conclusion that Enemies and Hordes starves most other rules (except two instances of Multi-way and only mildly two other Multi-way). Problematically as it is, we apply weights for FF3 to counter the multiple-occurrence starvation effect. The weights were calculated as the inverse function ($\frac{1}{x}$ when $x \neq 0$) of the average occurrence. The result for FF3 show positive effect for most of the meso-patterns (26 out of the 43 rules) except for the Gaps-, Enemy- and Horde-patterns for which the result, on the other hand, is absolute catastrophic (in table 2 the negative change is indicated in italic).

4 Expressive range

Smith & Whitehead [15] introduced the concept of *expressive range* of a level generator and suggested a set of possible metrics that illustrates diversity of the generated content. For PCG-tools it is interesting to show if the tool is able to generate content that is not identical. *Linearity* and *Leniency* were suggested as metrics for platform levels.

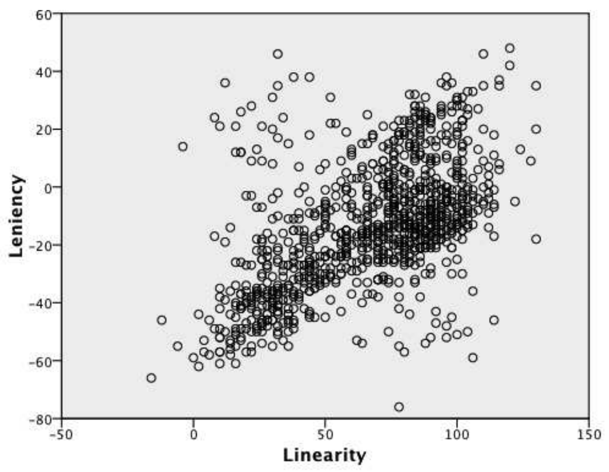

Fig. 3. The distribution of levels generated with FF1 on the two expressivity dimensions.

We have implemented versions of these metrics thus: Leniency is calculated across the whole level with +1 for gaps and enemies, and the reverse for the opposite −1 (for jumps with no gap associated, because jumps associated with danger is harder than jumps without danger). Linearity will be counted from the lowest point of the level, due to the fact that most micro patterns are connected to that and therefore all micro patterns forcing the player to jump due to a height difference of more than 1 tile will be considered as raising the non-linearity of the level.

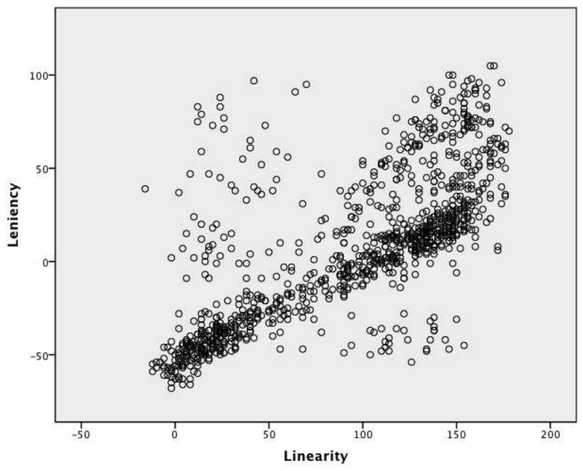

Fig. 4. The distribution of levels generated with FF2 on the two expressivity dimensions.

In figure 3, 4 and 5 we show a density plot based on the two metrics; leniency (LEN) and linearity (LIN) with 1000 generated levels for the fitness functions 1, 2 respectively 3 (FF1-3). FF1 have an expressive range in LEN of −75 to +50 with a concentration of levels around −20 to ±0 as well as an expressive range in LIN of −20 to +130 with a concentration in the range +50 to +100. FF2 gives LEN: −75 to +100 and LIN: −20 to +170. FF2 has two clusters; LEN/LIN −75 to −25/ ± 0 to +50 and −25 to 30/ + 85 to 160. Comparing the two fitness functions (FF1 & FF2) expressiveness yields that FF2 can generate both more difficult and more linear levels. The correlation that may exist is due to the gap and enemy placement in linear space in SMB (and in the micro-patterns) and it is more apparent due to the higher alignment to meso-patterns in FF2 than in FF1, which is more affected by the normal distribution in the variation of micro-patterns and get a less apparent cluster and range. FF3, however differ on all ranges; LEN: −105 to +80 & LIN −50 to +160. The two clusters; LEN/LIN: −100 to −30/−25 to +25 and −30 to +20/+50 to 130, are less apparent divided from each other and most of the individual members are not spread out as thin as before. The weighted fitness value gives a wider expressive range but the levels are more close if we observe the outliers suggesting that we could say that the *expressive spread* is affected with weighted patterns. The levels are more easy but also less linear. This is no surprise due to the low presence of meso-patterns of Gap-, Enemy- and Horde-type.

5 Discussion

Our approach could be viewed from a level designer's standpoint if we see the design process as handled by our three pattern levels; 1) at the micro-level,

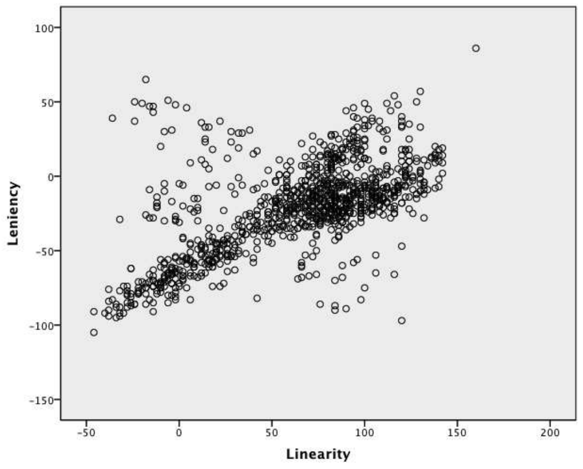

Fig. 5. The distribution of levels generated with FF3 on the two expressivity dimensions.

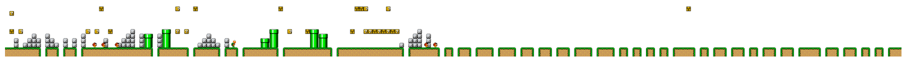

Fig. 6. An example of a generated level.

which contain the smallest representation level, in our approach the vertical slices function, 2) at the meso-level, where the combined slices in a certain order function to solve the challenges the designer wants to expose to the players to, and 3) at the macro-level handling the flow and overall (play-)experience of a level and/or game. If we implemented a planner that solved the issue of deciding on order of meso-patterns, difficulty (perhaps with the aid of metrics like leniency), training and educating the player, the full task of the level-designer, namely; to "... use a toolkit or 'level editor' to develop new missions, scenarios, or quests for the players. They lay out the components that appear on the level or map and work closely with the game designer to make these fit into the overall theme of the game" [18], could be solved for an entire game or genre.

In our fitness functions FF2 and FF3, we used weighted sums of the meso-pattern counters. There are well-known problems with fitness functions based on weighted sums, in particular that not all components are maximised at the same rate. An alternative would be to treat the problem as a multi-objective optimisation problem, and use specially designed evolutionary algorithms for this purpose. However, most such algorithms are designed for only a handful of objectives, which is problematic as our problem has dozens.

6 Conclusion

In this paper, we have introduced a pattern-based level generator for platform games. The general principle is to identify both micro-patterns and meso-patterns in the original game levels, represent new levels as combinations of micro-patterns and search for such combinations that express as many meso-patterns as possible. This way, micro-patterns are used as building blocks and meso-patterns as objectives. This principle, and the generator based on it, can easily be extended to a large range of different game types and game content types. To validate and explore the workings of our prototype level generator, we ran experiments with three different variations of our fitness function. We found that the generator could easily find certain patterns whereas others where harder to find, but that a rebalancing made it possible to find other patterns, sometimes at the cost of more frequent patterns.

7 Acknowledgments

We would like to thank Noor Shaker for the generated level image.

References

1. Shaker, N., Togelius, J., Nelson, M.J., eds.: Procedural Content Generation in Games: a Textbook and an Overview of Current Research. pcgbook.com (2013)
2. Togelius, J., Champandard, A.J., Lanzi, P.L., Mateas, M., Paiva, A., Preuss, M., Stanley, K.O.: Procedural content generation: Goals, challenges and actionable steps. In: Dagstuhl Seminar 12191: Artificial and Computational Intelligence in Games, Dagstuhl (2013)
3. Nintendo: Super Mario Bros. [Digital game] (1985)
4. Dahlskog, S., Togelius, J.: Patterns and Procedural Content Generation: Revisiting Mario in World 1 Level 1. In: Proceedings of the First Workshop on Design Patterns in Games. DPG '12, New York, NY, USA, ACM (2012) 1:1–1:8
5. Dahlskog, S., Togelius, J.: Patterns as Objectives for Level Generation. In: Proceedings of the Second Workshop on Design Patterns in Games. DPG '13 (2013)
6. Alexander, C., Ishikawa, S., Silverstein, M.: A pattern language – Towns, Buildings, Construction. Oxford University Press, New York, U.S.A. (1977)
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, U.S.A. (1994)
8. Björk, S., Holopainen, J.: Patterns in Game Design. Cengage Learning (2005)
9. Adams, E., Dormans, J.: Game Mechanics: Advanced Game Design. Voices That Matter. Pearson Education, Limited (2012)
10. Hullett, K., Whitehead, J.: Design Patterns in FPS Levels. In: FDG '10: Proceedings of the Fifth International Conference on the Foundations of Digital Games, New York, NY, USA, ACM (2010) 78–85
11. Smith, G., Anderson, R., Kopleck, B., Lindblad, Z., Scott, L., Wardell, A., Whitehead, J., Mateas, M.: Situating quests: design patterns for quest and level design in role-playing games. In: Proceedings of the 4th international conference on Interactive Digital Storytelling. ICIDS'11, Berlin, Heidelberg, Springer-Verlag (2011) 326–329

12. Cermak-Sassenrath, D.: Experiences with design patterns for oldschool action games. In: Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System. IE '12, New York, NY, USA, ACM (2012) 14:1–14:9
13. 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)
14. Togelius, J., Yannakakis, G., Stanley, K., Browne, C.: Search-based procedural content generation: A taxonomy and survey. Computational Intelligence and AI in Games, IEEE Transactions on **3**(3) (2011) 172–186
15. Smith, G., Whitehead, J.: Analyzing the expressive range of a level generator. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games. PCGames '10, New York, NY, USA, ACM (2010) 4:1–4:7
16. Shaker, N., Yannakakis, G., Togelius, J.: Crowdsourcing the aesthetics of platform games. Computational Intelligence and AI in Games, IEEE Transactions on **5**(3) (2013) 276–290
17. Karakovskiy, S., Togelius, J.: The mario ai benchmark and competitions. Computational Intelligence and AI in Games, IEEE Transactions on **4**(1) (2012) 55–67
18. Fullerton, T.: Game Design Workshop - A Playcentric Approach to Creating Innovative Games. Second edn. Morgan Kaufmann, New York, U.S.A. (2008)