A Multi-level Level Generator

Steve Dahlskog
Malmö University
Ö. Varvsgatan 11a
205 06 Malmö, Sweden
Email: steve.dahlskog@mah.se

Julian Togelius
IT University of Copenhagen
Rued Langaards Vej 7
2300 Copenhagen, Denmark
Email: julian@togelius.com

***Abstract*—Generating content at multiple levels of abstraction simultaneously is an open challenge in procedural content generation. Representing and automatically replicating the style of a human designer is another. This paper addresses both of these challenges through extending a previously devised methodology for pattern-based level generation. This method builds on an analysis of Super Mario Bros levels into three abstraction levels: micro-, meso- and macro-patterns. Micro-patterns are then used as building blocks in a search-based PCG approach that searches for macro-patterns, which are defined as combinations of meso-patterns. Results show that we can successfully generate levels that replicate the macro-patterns of selected input levels, and we argue that this constitutes an approach to automatically analysing and replicating style in level design.**

I. Introduction

Procedural content generation in games (PCG) refers to the algorithmic creation of game content, with no or limited human input. Recent years has seen a marked increase in interest in PCG in the game development community, where it is now routinely used both for runtime level generation in certain types of games (e.g. rogue-likes) and for offline generation of certain types of content, such as vegetation and terrains. This development is paralleled by a significant increase in PCG research in academia. Unlike in commercial game development, the focus tends to be on more ambitious forms of PCG than what is currently seen in released games, and using more complex methods [1].

In a recent overview paper, a number of long-term goals and research challenges for PCG are described [2]. The paper suggests the following grand goals of PCG: *Multi-level Multi-content PCG*, *PCG-based Game Design* and *Generating Complete Games*. It is argued that work addressing any of its nine more concrete research challenges would contribute to progress towards realising these grand goals of PCG. Further, five very concrete actionable steps are listed, each of which is envisioned to address one or several of the research challenges.

In this paper, we address two of the research challenges, namely *Representing Style* and *General Content Generators*, and one of the actionable items, namely *Competent Mario Levels*. Representing Style refers to being able to create a generative model of the style of a particular designer or a particular design school, whereas General Content Generators refers to being able to generate either different types of content (on different levels of abstraction) for a single game or content for multiple games. The Competent Mario Levels actionable step refers to creating level generators for Super Mario Bros that can create varied, interesting, good-looking, playable and entertaining levels.

The way we address these challenges is to extend an existing pattern-based level generator for Super Mario Bros. In previous work, we have described a method which builds levels for Super Mario Bros out of "micro-patterns", i.e. thin level slices, and uses an evolutionary algorithm to search for levels that contain multiple instances of "meso-patterns", which are larger designed structures [3, 4, 5]. It was observed that while this method generated playable levels with interesting micro-structure, the levels lacked a sense of progression, unity or other macroscopic properties. The working hypothesis of this paper is that such macroscopic structure can be achieved with an extension of this method by using objectives at a higher abstraction level. This in turn requires that such objectives can be extracted from existing game levels.

A. Contributions in this paper

In previous work, we have identified meso-patterns in Super Mario Bros [3], and devised a search-based approach to level generation in the Mario AI Benchmark where micro-patterns are used as building blocks and meso-patterns as objectives [4, 5]. In this paper, we introduce a third level of abstraction, macro-patterns, defined as the occurrence and sequence of meso-patterns. We also describe a level analyser, which extracts patterns from existing levels. Finally, we describe the results of experiments in evolving levels using macro-patterns as objectives. For this purpose we have also devised a new mutation operator for game levels based on cutting and pasting micro-patterns.

II. Background

Our work builds on previous work in both design-oriented and technical game research. Here, we describe previous work on PCG in games, design patterns, and the combinations of these, and we also present the benchmark game used for the experiments.

A. Procedural content generation in games

Game content refers to any game asset excluding non-player character (NPC) behaviour and the game engine - for example levels, rules, textures, narrative and in-game items. PCG has recently attracted considerable interest in the digital game research community, as evidenced by hundreds of publications and the establishment of a dedicated workshop running annually since 2010. This is at least partly due to there

being multiple good reasons to attempt to create algorithms that generate content, including: reducing the cost and time of game development, enabling infinite and/or adaptive games, studying game design by formalising human creativity, and attempting to surpass such creativity. In the current context, we are interested both in the computational study of game design, and in creating fast algorithms that can reliably supply a game with large amounts of quality content.

The last few years has seen a surge of interest in an approach to PCG called search-based PCG, where evolutionary algorithms or other global stochastic optimisation algorithms are used to generate content [6]. The two most important considerations here are content representation (how the genotype, e.g. levels, is represented as a phenotype, e.g. vectors of integers, on which the variation operators work) and content evaluation (how a fitness value is assigned to a content artefact).

B. Design Patterns

Design patterns were initially proposed by Alexander [7], an architect who created them with the intent to empower individuals to express their ability to design. Design patterns are basically a rather informal grammar containing a set of descriptions covering reoccurring design problems in a domain. This problem description is paired with a suggested core solution which could be reused. In effect, the second of the two components (problem & solution) is very versatile due to the generalisation of the solution space. Design patterns have been adopted in object-oriented analysis and design [8], and thinking of software architecture in terms of design pattern solutions has become very influential. Björk and Holopainen later applied the ideas of design pattern to game design, listing hundreds of generic game design patterns in an influential book [9, 10]. Several authors have further identified a number of game design patterns in specific game genres [11, 12, 13, 14].

In the current paper we are principally interested in patterns in *level design*, where levels are the structures that the player character traverses (not to be confused with e.g. levels of abstraction). Given the fact that games often are designed artefacts put together with a purpose, several aspects of them can be viewed as structures. For instance, rules govern the process of play, whereas levels and game space is often indirectly controlling the movement of the player, and objects in games usually have a specific purpose effectively limiting their use for the player. In relation to this, we could view the content in a platform game (including levels) as structured design objects, i.e. objects following design patterns.

C. Benchmark game

In this paper we will use the game *Super Mario Bros.* (SMB) [15] as a benchmark. The game was first released by Nintendo in 1985, and is a side scrolling 2-dimensional "platformer" game. SMB has become very influential through setting a number of standards for the platformer genre, and has helped bring about the genre's popularity. In the game, the protagonist Mario (or his brother Luigi) moves from left to right, jumping onto platforms or other structures to overcome obstacles or onto enemies to squash them. SMB consists of 8 worlds, each containing 4 levels, where the three first levels span from a starting point (left-most) to the end by a castle entrance (right-most) and the fourth level ends in a "boss-fight". As there is no interface for NPC control or level generation in the original game, we build on the *Mario AI Framework*, a software toolkit which was developed for the Mario AI Competition [16, 17, 18]. This software is based on *Infinite Mario Bros*, a clone of SMB that focused on the non-"boss-fight"-levels. In SMB the levels have a varied length of 148 to 377 with an average of 200 tiles. Various approaches to generate levels for the Mario AI Framework have been proposed, as surveyed in [17, 19]; approaches that explicitly copy the style of SMB levels include Markov chains [20].

TABLE I. PATTERNS FOR SUPER MARIO BROS. GROUPED BY THEME [3].

Enemies	
Enemy	A single enemy
2-Horde	Two enemies together
3-Horde	Three enemies together
4-Horde	Four enemies together
Roof	Enemies underneath a hanging platform making Mario bounce in the ceiling
Gaps	
Gaps	Single gap in the ground/platform
Multiple gaps	More than one gap with fixed platforms in between
Variable gaps	Gap and platform width is variable
Gap enemy	Enemies in the air above gaps
Pillar gap	Pillar (pipes or blocks) are placed on platforms between gaps
Valleys	
Valley	A valley created by using vertically stacked blocks or pipes but without Piranha plant(s)
Pipe valley	A valley with pipes and Piranha plant(s)
Empty valley	A valley without enemies
Enemy valley	A valley with enemies
Roof valley	A valley with enemies and a roof making Mario bounce in the ceiling
Multiple paths	
2-Path	A hanging platform allowing Mario to choose different paths
3-Path	2 hanging platforms allowing Mario to choose different paths
Risk and Reward	A multiple path where one path have a reward and a gap or enemy making it risky to go for the reward
Stairs	
Stair up	A stair going up
Stair down	A stair going down
Empty stair valley	A valley between a stair up and a stair down without enemies
Enemy stair valley	A valley between a stair up and a stair down with enemies
Gap stair valley	A valley between a stair up and a stair down with gap in the middle

III. LEVEL DESIGN PATTERNS IN MARIO

We have previously analysed the content of the original game with the aid of a framework for 2D Platformer games [21] and heuristics for playability [22] and suggested a set of (meso-) patterns that SMB levels consists of [3]. We identified patterns on two levels, micro-patterns and meso-patterns. Micro-patterns are simply vertical slices of the level. Meso-patterns are features such as groups of *enemies*, *gaps* to jump over, *valleys* boxing in parts of the level, allowing the player to choose *multiple paths* and elevating Mario with the aid of *stairs*. In this paper, we also introduce macro-patterns, which are sequences of meso-patterns.

A. Micro-patterns

The content in SMB can be viewed from Mario's standpoint, namely, horizontally from left to right one tile at the

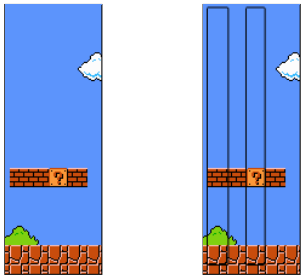

Fig. 1. To the left we have a excerpt from SMB World 1–Level 1 which can be replicated with only two micro-patterns (slices) marked with black frames to the right. It also exemplifies a 2-Path pattern.

moment. If one imagine the levels as one tile wide slices and collect them in a library, the first level, even though it is 199 slices "long" only 27 different slices are used. These slices could be viewed as micro-patterns since they, in themselves, are designed content, and they often contain several pieces put together like a Goomba, a question-mark-box, a brick-block or something else that is either an obstacle, or aid to the player. In fig 1, the left-most slice or micro-pattern contain mostly empty space (allowing Mario to jump) but also to land on a Goomba or a ground-tile. If Mario were to walk into this slice the player either loses a life or a power-up effect. These micro-patterns works in a similar way as the tiles when decomposing the problem of generating dungeons [23].

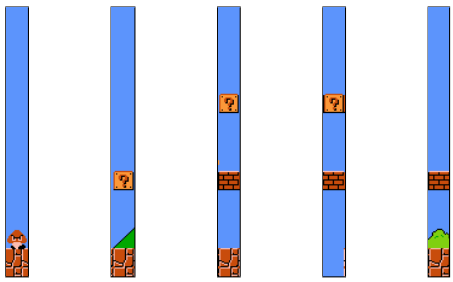

Fig. 2. Examples of similar but still unique slices. The two to the right can be used to create the structure of Fig. 1 and a sub-set of them can be used to create most of Fig. 4

B. Meso-patterns

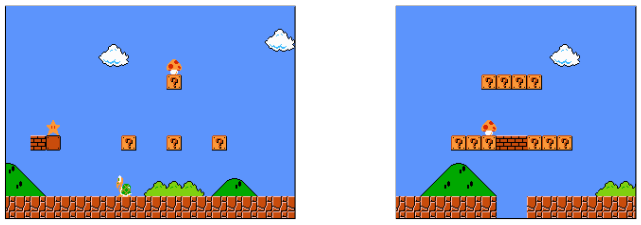

Fig. 3. Two meso-patterns (to the left; a sparse *Risk and Reward* (W1L1) and to the right a dense *3-Path* (W4L1).

The next meaningful level are the meso-patterns. These are perhaps best explained as instances of the different patterns previously identified [3]. A meso-pattern is a feature which requires or affords some player action, like three Goombas walking in a single file formation on ground. If the micro-patterns are to be seen from Mario's viewpoint, the meso-patterns could be viewed more like how the player sees the content: sequences of slices making up most of a screen.

It is important to note that meso-patterns are a bit more abstract than micro-patterns; each meso-pattern could be instantiated in multiple ways, by different configurations of micro-patterns. For example, a valley could consist of 5 or 10 slices, but still be a valley. If our micro-patterns are identified by an integer then a meso-pattern is a string of specific integers like $5-1-1-7$ or $1-8-8-5-8-1$[1]. (These particular strings are taken from the original game.)

C. Macro-patterns

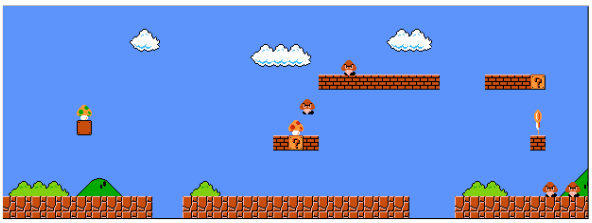

Fig. 4. A Macro-pattern example from SMB, stretching over two screens, where a *2-Path* and a *Gap* continues on to a *Risk and Reward* and a *Gap* onto a *3-Path* with an end consisting of a *2-Horde*.

The meso-patterns are helpful to understand the content of SMB but does not convey more macroscopic level structure. For that we suggest a higher level – the macro-pattern level. On this level the relation between different meso-patterns becomes clear and the placement of individual power-up-mushrooms can balance difficulties that lies beyond the current screen. In figure 4 we can see an example of how patterns are connected together over more than one screen. At this level of abstraction the level designer can provide the player with a greater play experience by providing a steady and controlled difficulty curve, teach the player how to tackle new obstacles and enemies. "Pedagogic" macro-patterns, where a meso-pattern first appears in a simpler form and then in a more complex form, so that the player can first overcome the simpler challenge to then be ready to face a harder challenge of the same type, are common among the original SMB levels. Story arcs could be partly implemented, or at least supported, on the macro-level as well. Given this, it might be possible to argue for further abstraction levels covering the "Worlds" of SMB or perhaps the full game or even the whole game franchise.

For example, if we describe the first level of SMB, i.e. World 1–Level 1 (W1L1, see Fig. 5) as a sequence of meso-patterns we get the following: *Risk and Reward*, *(Empty) Pipe valley*, *(2-Horde) Pipe Valley*, *2-Path*, *Gap*, *Risk and Reward*, *2-Horde*, *Risk and Reward*, *Risk and Reward*, *(Empty) Stair valley*, *(Gap) Stair valley*, *Roof valley*, *Stair up*.

D. Multi-Level Level Generation

In our suggested approach we utilise a "bottom-up"-approach where the micro-level is the foundations for the meso-level which in turn makes up the macro-level. Our method is a *search-based* PCG approach [6], described below.

[1]The last string is for instance seen in Fig. 1

Fig. 5. Level 1, World 1 from the original Super Mario Bros game, reimplemented in the Mario AI Framework (SMB-W1–L1).

Fig. 6. Level 1, World 8 (SMB-W8–L1) (mid 200 tiles, start and ending empty ground is cropped).

IV. Pattern-based level generation

We have previously presented two level generators for the Mario AI Framework that builds on the identified patterns. The first of these was a simple constructive pattern-based level generator that combined pre-fabricated instances with minor variations depending on assigned parameters on difficulty and reward settings [3]. The second generator takes a search-based approach, with a representation based on micro-patterns and objective function based on the existence and number of meso-patterns [4]. Two versions of the fitness function were developed: one which simply counted every occurrence of every meso-pattern, and one which only counted the number of individual meso-patterns that could be found in the level. It was found that levels that scored highly on either of these metrics were perceived as better-designed than those that scored lower, but also that those that were only optimised for the first variation (every occurrence) became rather dense. After further experimenting [5] with its multi-objective fitness functions, we here extend it to cover the macro-pattern level as well.

V. Automatic level analysis

In order to be able to generate levels that replicate the sequence of meso-patterns from existing levels, we first need to be able to extract this sequence. For this purpose we built a level analyser. The level analyser takes any Super Mario Bros (or Infinite Mario Bros) level encoded in a specific simple file format and returns a list of all the micro-patterns (slices) in the level and their frequencies, and the order of all meso-patterns. This is technically an array of integers where each integer represents a particular meso-pattern out of those identified in [3], but can be read out as e.g. "{pipe-valley, three-horde, three-horde, stair}" etc. The same pattern detection code is used here as is used in the objective functions.

VI. Methods

In this section we stepwise go through our approach, by stating the principal parts; representation, algorithm and fitness function.

A. Representation

Our level generator output is a single SMB level with the length of 200 and a height of 14 tiles. The internal representation of a level is an array of integers, where each integer represents a micro-pattern (see Fig. 2 for examples).

B. Evolutionary Algorithm

Our search-based approach uses a fitness function that rewards the presence of meso-patterns with a simple $\mu + \lambda$ evolution strategy where $\mu = \lambda = 100$ combined with the operators *single-point mutation* and *one-point crossover*. This means, when we use a population of 200 members, that we discard the 100 members with lowest fitness and use the best 100 members as parents for breeding pairwise. All of the newly generated offspring are also subject to mutation. We consequently deem members with unplayable content as unfit for breeding by setting their fitness value extremely low.

C. Variation operators

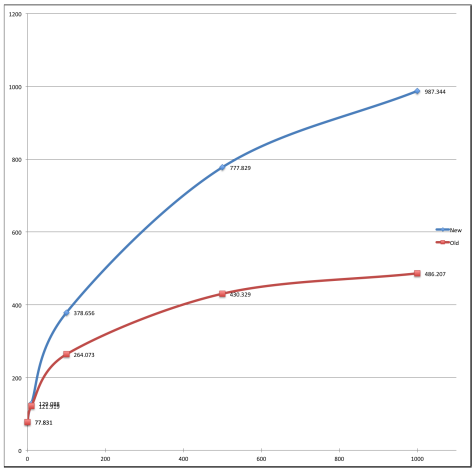

Fig. 7. A comparison between the effect of the mutation-operators.

In previous work our mutation operator simply exchanged a single micro-pattern for another, randomly selected [5]. Given the relative length of a micro-pattern (1 slice = 1 block), in relation to a full level in SMB (148-377 slices) and the nature of our initial mutation operator; exchanging a single slice for another for the whole member (meaning a mutation effect of 0.5% out of 200 slices) we opted to incorporate a more aggressive mutation (blue line in Fig. 7). Instead of the minimalistic mutation operator working as an exchange of a single slice exchange (red line) we apply a sequence exchange. The new mutation operator change a set of five slices at a random starting position with a new random set of five slices.

D. Fitness functions

Our fitness functions measure the presence and order of patterns and are based on string search. A fitness value is

assigned to each level based on the presence of specific sub-strings representing meso-patterns taken from SMB. A sub-string is typically seen in Fig. 1 made up with micro-patterns (see Fig. 4). Since these sub-strings vary in length and complexity some patterns are harder to find in the solution space than others. This, in turn, yields that we need to understand how to define the fitness function according to the wanted outcome. We focus our attention on the difference between finding all patterns we have defined in solution space (with the fitness function *FFMeso*) without rewarding any specific pattern over another. From there we utilise a weighted value based on previous experiments [5] (called *FFMesoB[alanced]*) in order to understand the effect of the added macro-level works in the solution space (called *FFMacro*). *FFMacro* is based on a relative reward value so that it rewards the correct order of meso-patterns according to the original SMB meso-pattern order in addition to how FFMesoB reward sub-strings. In short, if the order of meso-patterns in a member corresponds to the one in the target level it is more probable that it is chosen for breeding. Note that we are only looking for instances of meso-patterns and not the exactly same pattern implementations as in SMB.

VII. Results

Our experiments are evaluated in three ways: (1) We measure the meso-pattern (type and how many) for the best member of a 1000 generation search (200 members with the length of 200 micro-patterns); (2) we compare the fitness values distribution for the fitness functions; and (3) we apply *expressive range* analysis (see section VII-B).

In order to get some input on diversity aspects of the different fitness functions we have generated 100 levels for each fitness function and compare them to each other. FFMeso favours simple patterns like enemies and hordes and seldom provide anything more complex like multi-way and pipes. FFMesoB and FFMacro provide better overall coverage of patterns. FFMesoB and FFMacro does not differentiate very much but generally FFMacro provide some improvements on longer patterns (indicated in italic in table II).

FFMeso generally perform uniform values. It should be noted that since this fitness value favours low ranked rewards and is then compared to the weighted macro fitness function very little variation is gained (see Table II to see the pattern distribution) it should not be directly compared value by value with the other two fitness functions which are more compatible in regard to comparison. In that aspect both fitness functions can generate macro-pattern ordered in a level but FFMacro perform a bit better reaching a macro-pattern fulfilment of a maximum $7/12$ and a common level of $4/12$ whereas FFMesoB only reaches $6/12$ (see Table III). FFMesoB has a higher maximum altogether because it can fit in more high value patterns in the level. FFMacro tries to find the right pattern there which could be improved by rewarding macro pattern order more, but of course then running the risk of starving the meso-patterns altogether.

Figures 11 show a number of generated levels for visual comparison. These were all generated with level 1-1 (as seen in figure 5) as target level. It can be seen from these pictures that the levels generated with the Macro fitness function appear to have more large-scale structure, or at least more variation on the macro scale.

A. Efficiency

FFMeso and FFMesoB can run in fair online environments generating a level with the length of 200 based on a 200 member population and 1000 generations in 4 seconds with the current implementation in Java running in NetBeans IDE on a 2011 MacBook Pro. However, the FFMacro, have to account for relative reward values and an extra data structure (that keeps track of the order in relation to the wanted order) which affects execution time tenfold effectively placing this approach in the offline PCG application range.

B. Expressive Range

The concept of *expressive range* could be seen as the approach to visualise and measure the variation of the generated content according to a representative metric [24, 19]. This would allow understanding the diversity and uniqueness of a level generator. In our case we will apply Smith's & Whitehead's metrics *Linearity* and *Leniency* [24].

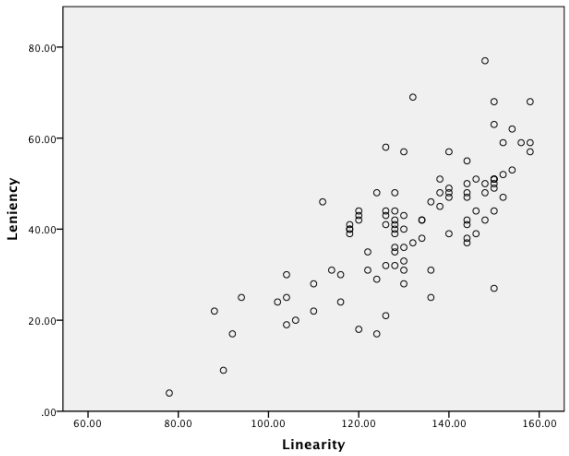

Fig. 8. The distribution of levels generated with FFMeso on the two expressivity dimensions.

Our implementation works as follows; Leniency is calculated across the whole level with $+1$ for gaps and enemies, and -1 for jumps without danger. Linearity will be counted as $+1$ for any change from the floor of the level, due to the fact that most micro patterns is connected to that. In Fig. 8 the output of FFMeso and in Fig. 9 the output of FFMacro are displayed using 100 unique levels from the two different fitness function used.

Comparing the FFMacro and FFMeso we can see that they occupy a different expressive space with FFMeso generating levels more similar internally than the other two fitness functions. FFMeso has a linearity range of 80 and leniency range of 80 whereas FFMacro has 75 and 105 for linearity and leniency respectively. Comparing their (FFMeso and FFMacro) individual space we can see there is very little overlap in their expressive range.

Given this (see Fig. 10) we can conclude that the *Linearity* of the FFMesoB and FFMacro are more variated but also

TABLE II. FOUND PATTERNS (RULES) IN FFMESO, FFMESOB AND FFMACRO BASED ON 100 LEVELS AND 1000 GENERATIONS PER LEVEL.

Pattern	Mesa		Straight	Multi-way										
Occurrence in FFMeso	16	11	*132*	4	3	3	1	0	3	2	4	1	2	1
Average in FFMeso	0.16	0.11	1.32	0.04	0.03	0.03	0.01	0	0.03	0.02	0.04	0.01	0.02	0.01
Occurrence in FFMesoB	*50*	50	61	131	*39*	7	2	*90*	*39*	*39*	131	141	51	53
Average in FFMesoB	0.5	0.5	0.61	1.31	0.39	0.07	0.02	0.9	0.39	0.39	1.31	1.41	0.51	0.53
Occurrence in FFMacro	28	*55*	57	*137*	39	*8*	*3*	90	39	38	*137*	*143*	*62*	*62*
Average in FFMacro	0.28	0.55	0.57	1.37	0.39	0.08	0.03	0.90	0.39	0.38	1.37	1.43	0.62	0.62

Pattern	Enemy						Hordes					Gaps			
Occurrence in FFMeso	2129	1510	221	2354	1310	97	553	753	1380	1380	545	4	6	2	11
Average in FFMeso	21.29	15.10	2.21	23.54	13.10	0.97	5.53	7.53	13.80	13.80	5.45	0.04	0.06	0.02	0.11
Occurrence in FFMesoB	34	3	1	29	3	1	0	1	0	0	0	*112*	22	11	25
Average in FFMesoB	0.34	0.03	0.01	0.29	0.03	0.01	0	0.01	0	0	0	0.112	0.22	0.11	0.25
Occurrence in FFMacro	32	*2*	*0*	*40*	*5*	*2*	*0*	*3*	*4*	*4*	*0*	103	21	*17*	24
Average in FFMacro	0.32	0.02	0	0.4	0.05	0.02	0	0.03	0.04	0.04	0	1.03	0.21	0.17	0.24

Pattern	Valley			Stair					Pipes					
Occurrence in FFMeso	0	0	1	18	21	18	10	18	0	0	0	0	0	1
Average in FFMeso	0	0	0.01	0.18	0.21	0.18	0.1	0.18	0	0	0	0	0	0.01
Occurrence in FFMesoB	35	72	53	97	94	133	139	134	5	20	39	19	231	111
Average in FFMesoB	0.35	0.72	0.53	0.97	0.94	1.33	1.39	1.34	0.05	0.2	0.39	0.19	2.31	1.11
Occurence in FFMacro	38	90	86	79	89	129	102	90	7	19	40	17	223	117
Average in FFMacro	0.38	0.9	0.86	0.79	0.89	1.29	1.02	0.9	0.07	0.19	0.4	0.17	2.23	1.17

TABLE III. COMPARISON OF FOUND MACRO PATTERNS

	MIN	MAX	MEAN	DEV	No. 0	No. 1	No. 2	No. 3	No. 4	No. 5	No. 6	No. 7
FFMesoB	0	6	2.6	1.52	14	11	12	37	19	4	3	0
FFMacro	0	7	3.2	1.53	7	10	9	24	36	9	4	1

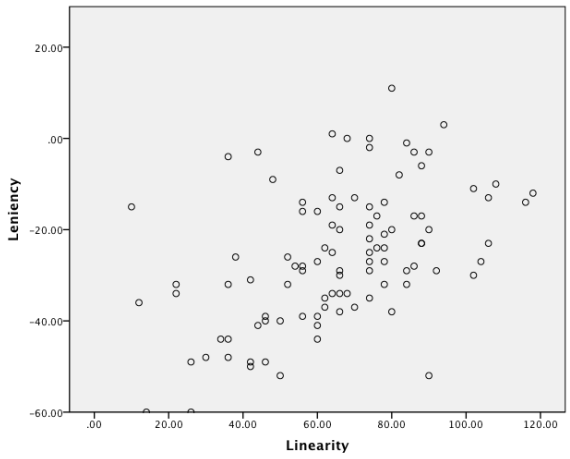

Fig. 9. The distribution of levels generated with FFMacro on the two expressivity dimensions.

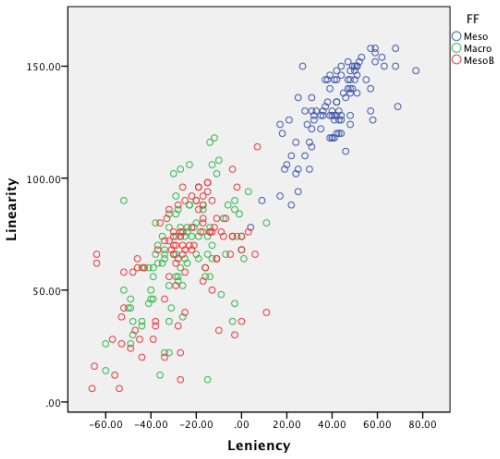

Fig. 10. The distribution of levels generated with FFMeso, FFMesoB and FFMacro on the two expressivity dimensions.

less hard to complete (probably due to the lower number of enemies present in these levels see table. II). However, since the distribution of different patterns are more like the original game SMB the FFMesoB and FFMacro are probably more interesting for a player.

VIII. FUTURE WORK

Given the set of generators available to the PCG-research-community more in dept studies of the diversity of the different approaches would yield welcome knowledge. For instance, how well does the generator fulfil the intended goal and how does that relate to other generators abilities. Can we, with the use of metrics or empirical tests order the different generators on a spectra ranging from variation to control? Does implementation of different but similar techniques place generators close to each other on that spectra?

Considering the multi-level search-based and bottom-up-approach applied in this paper it would be interesting to compare it with other possible approaches for multi-level generators. Especially, top-down and constructive approaches would be a welcome comparison. The top-down approach could function in different ways, ranging from a more automated version were the user supplied parameters and the generator suggested levels to a more user centred approach where the designer marked our space in a level and picked pattern definitions and placed them in an order suited to the designer and mixing designer work with constraints on the generator to fulfilling patterns and even down to level where the designer defines new meso- and micro-patterns.

IX. CONCLUSION

In this paper we have suggested a search-based PCG method and level generator for platform games that incorporates three levels of patterns, namely; 1) micro-, 2) meso- and 3) macro-patterns. These three levels handles different aspects of the level generation ranging from low level detail

to full level overview. To demonstrate the effect the multi-level level generator we ran a set of experiments with three different fitness functions; FFMeso (rewarding meso-patterns), FFMesoB (a balanced version using weights derived from a previous version [5]) and FFMacro (using the same weights but with an added extra reward if the order of the patterns we in alignment to the original SMB game). During this exploration of the solution space we noted that some patterns are affecting the presence of other patterns and that the expressive range can vary based on the used fitness function. The added macro-level have increased the run-time of the level generator tenfold making the generator more suitable for offline generation rather than online.

REFERENCES

[1] J. Togelius, N. Shaker, and M. J. Nelson, "Introduction," in *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*, N. Shaker, J. Togelius, and M. J. Nelson, Eds. Springer, 2014.

[2] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley, "Procedural content generation: Goals, challenges and actionable steps," in *Dagstuhl Seminar 12191: Artificial and Computational Intelligence in Games*. Dagstuhl, 2013.

[3] S. Dahlskog and J. Togelius, "Patterns and Procedural Content Generation: Revisiting Mario in World 1 Level 1," in *Proceedings of the First Workshop on Design Patterns in Games*, ser. DPG '12. New York, NY, USA: ACM, 2012, pp. 1:1–1:8.

[4] ——, "Patterns as Objectives for Level Generation," in *Proceedings of the Second Workshop on Design Patterns in Games*, ser. DPG '13, May 2013.

[5] ——, "Procedural Content Generation Using Patterns as Objectives," in *Proceedings of EvoGames, part of EvoStar.*, A. I. Esparcia-Alcázar, Ed., 2014.

[6] J. Togelius, G. N. Yannakakis, K. O. Stanley, and C. Browne, "Search-based procedural content generation: A taxonomy and survey," *IEEE Transactions on Computational Intelligence and Games*, vol. 3, pp. 172–186, 2011.

[7] C. Alexander, S. Ishikawa, and M. Silverstein, *A pattern language – Towns, Buildings, Construction*. New York, U.S.A.: Oxford University Press, 1977.

[8] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, U.S.A.: Addison-Wesley, 1994.

[9] S. Björk and J. Holopainen, *Patterns in Game Design*. Cengage Learning, 2005.

[10] S. Björk, "Game Design Patterns 2.0," Web page, March 2013. [Online]. Available: http://gdp2.tii.se/

[11] K. Hullett and J. Whitehead, "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, pp. 78–85.

[12] D. Cermak-Sassenrath, "Experiences with design patterns for oldschool action games," in *Proceedings of The 8th Australasian Conference on Interactive Entertainment: Playing the System*, ser. IE '12. New York, NY, USA: ACM, 2012, pp. 14:1–14:9.

[13] C. Lewis, N. Wardrip-Fruin, and J. Whitehead, "Motivational game design patterns of 'ville games," in *Proceedings of the International Conference on the Foundations of Digital Games*, ser. FDG '12. New York, NY, USA: ACM, 2012, pp. 172–179.

[14] C. Dristig Stenström and S. Björk, "Understanding Combat Design in Computer Role-Playing Games," in *Proceedings of the Second Workshop on Design Patterns in Games*, ser. DPG '13, May 2013.

[15] Nintendo, "Super Mario Bros." [Digital game], 1985.

[16] J. Togelius, S. Karakovskiy, and R. Baumgarten, "The 2009 Mario AI Competition," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC)*, 2010.

[17] N. Shaker, J. Togelius, G. N. Yannakakis, B. G. Weber, T. Shimizu, T. Hashiyama, N. Sorenson, P. Pasquier, P. A. Mawhorter, G. Takahashi, G. Smith, and R. Baumgarten, "The 2010 mario ai championship: Level generation track," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 3, no. 4, pp. 332–347, 2011.

[18] S. Karakovskiy and J. Togelius, "The Mario AI Benchmark and Competitions," *Computational Intelligence and AI in Games, IEEE Transactions on*, vol. 4, no. 1, pp. 55–67, 2012.

[19] B. Horn, S. Dahlskog, N. Shaker, G. Smith, and J. Togelius, "A comparative evaluation of procedural level generators in the mario ai framework," in *Proceedings of the 9th International Conference on Foundations of Digital Games*, ser. FDG '14, 2014.

[20] S. Snodgrass and S. Ontañón, "Experiments in map generation using markov chains," in *Proceedings of the 9th International Conference on Foundations of Digital Games*, ser. FDG '14, 2014.

[21] G. Smith, M. Cha, and J. Whitehead, "A Framework for Analysis of 2D Platformer Levels," in *Sandbox '08: Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*. New York, NY, USA: ACM, 2008, pp. 75–80.

[22] H. Desurvire, M. Caplan, and J. Toth, "Using Heuristics to Evaluate the Playability of Games," in *CHI 2004 Extended Abstracts on Human Factors in Computing Systems*, April 2004.

[23] C. McGuinness and D. Ashlock, "Decomposing the level generation problem with tiles," in *IEEE Congress on Evolutionary Computation*. IEEE, 2011, pp. 849–856.

[24] G. Smith and J. Whitehead, "Analyzing the expressive range of a level generator," in *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, ser. PCGames '10. New York, NY, USA: ACM, 2010, pp. 4:1–4:7.

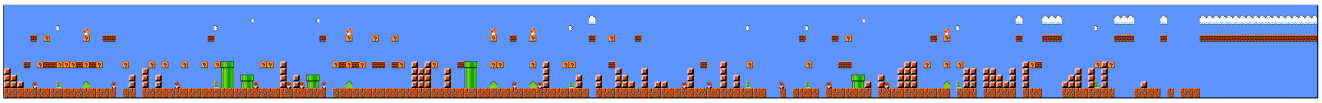

Fig. 11. FFMacro #26 MC: 4, fitness value: 441 (lowest).

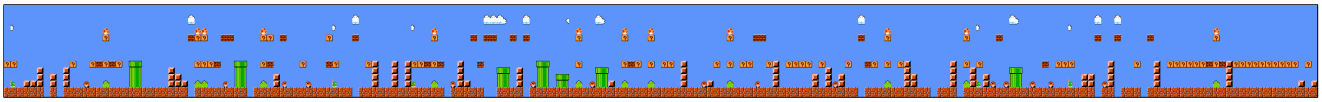

Fig. 12. FFMacro #28 MC: 6, fitness value: 1315.

Fig. 13. FFMacro #35 MC: 0, fitness value: 850.

Fig. 14. FFMacro #82 MC: 7, fitness value: 1485.

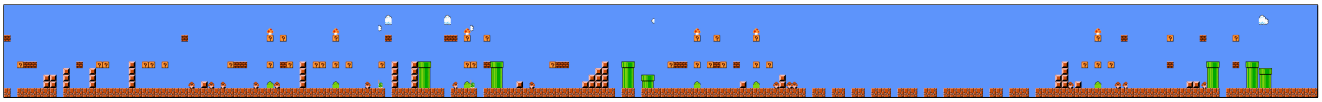

Fig. 15. FFMacro #98 MC: 6, fitness value: 2332 (highest).

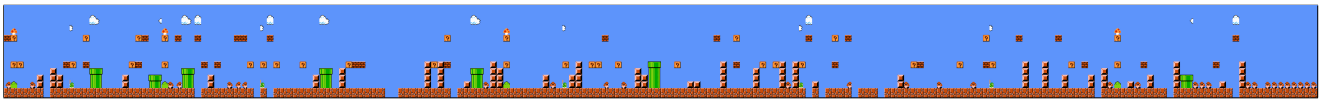

Fig. 16. FFMesoB #6 MC: 0, fitness value: 545.

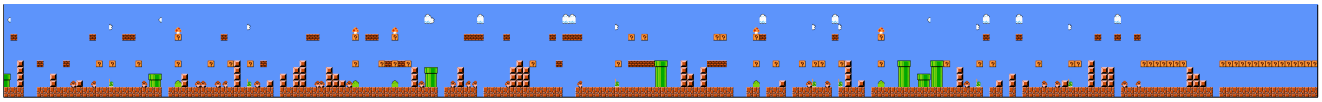

Fig. 17. FFMesoB #30 MC: 3, fitness value: 409 (lowest).

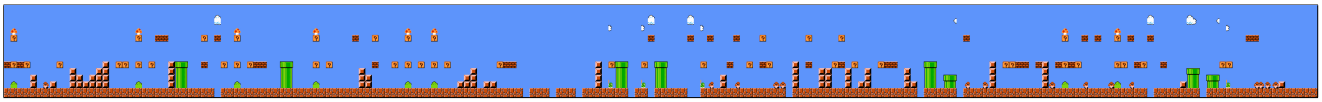

Fig. 18. FFMesoB #64 MC: 6, fitness value: 2065 (highest).

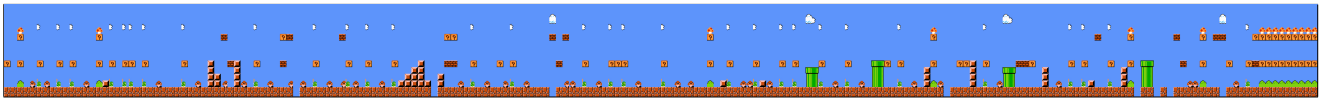

Fig. 19. FFMeso #42 MC: 2, fitness value: 202 (highest).

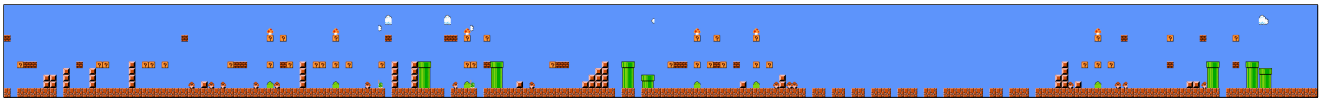

Fig. 20. FFMeso #98 MC: 0, fitness value: -47 (lowest).