Patterns and Procedural Content Generation

Revisiting Mario in World 1 Level 1

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

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

ABSTRACT

Procedural content generation and design patterns could potentially be combined in several different ways in game design. This paper discusses how to combine the two, using automatic platform game level design as an example. The paper also present work towards a pattern-based level generator for Super Mario Bros, namely an analysis of the levels of the original Super Mario Bros game into 23 different patterns.

Categories and Subject Descriptors

K.8.0 [**Personal Computing**]: General-Games

Keywords

Procedural Content Generation, patterns, design, platform games, Super Mario Bros.

1. INTRODUCTION

This paper discusses the relation between procedural content generation (PCG) and design patterns in games, and presents work-in-progress on a design pattern-based level generator for *Super Mario Bros.* (SMB). Procedural content generation in digital games refers to the automated or semi-automated creation of game content using algorithms. Design patterns are a way of structuring design and the design process into recurring elements. This way of thinking originated in architecture, has had major impact on software development and has recently been applied to game design.

In the following, we will first discuss procedural content generation and in particular its role in providing variation in game. We will then discuss design patterns, and how they can (and have been) used for PCG. We then describe ongoing work with identifying and describing level design patterns in *SMB*, and building a level generator based on these patterns.

1.1 Procedural content generation

DPG 2012 May 29, 2012, Raleigh, North Carolina, USA

In the digital game industry the PCG has successfully been used for the creation of variations of content to enable replay of a digital game. The types of game content that have been generated range from game worlds, levels and items to ornamental decoration. Examples of generation of complete game worlds include *Terraria* [17] and *Civilization* [14]. *Diablo* [4], *Borderlands* [10] and the *SpeedTree* [12] middleware are examples of generation of levels, items and decoration, respectively. An interesting example of PCG being used as a form of data compression is the space trading game *Elite* [1] where David Braben and Ian Bell succeeded to squeeze in 8 galaxies with 256 stars each into the limited (22 kB) memory capacity of the BBC Microcomputer with the help of pseudo-random numbers [21].

1.2 Structures, noise and meaning

Games are in many aspects a combination of designed structures. Rules govern the game's use in the play process in a way that creates meaning to players by allowing or disallowing actions. Levels guide the player through the game world, and sometimes, game space as well, because of its way of structuring challenges and rewards in the game space. The game's story builds meaning to the actions of non-player characters (NPCs) and other entities populating the game world and together with quests, the story provide purpose to the challenges the player is presented with. Thus we conclude that structures in games are fundamentally entities that are inter-connected with each other by relationships in a stable and consistent matter for a single game. They should also be observable and recognizable from the user's perspective, either for the player character (PC), on the player's experience level or beyond the game, like game genres and themes. It is from the different structures we define meaning and consistency in abstract things as digital games.

"Structure (in [...] games) operates much like context, and participates in the meaning-making process. By ordering the elements of a system in very particular ways, structure works to create meaning." [18]

Typically the designed structure is a functional one if helps to create meaning for the player. By analyzing the structures in games we can find the reason to why a certain game makes meaning in a certain way. Digital games are with few exceptions (e.g. *Dark Room Sex Game* [7]) relaying on its ability to convey information on the game state through visual output. In action games, more often than not, the player needs to scan, interpret and assess the information of the game state at high speed. The assessment of the game

state allows the player to interact with the game. In design practice the user's ability to act is conveyed through affordances and the user's inability to act is conveyed through constraints [16]. In order to be able to procedurally generate suitable content for a digital game the algorithm has to be able to produce output that is meaningful to the player. Unless the player is able to understand the output's meaning it will be experienced as noise or make the player chose a less advantageous or perhaps even make the player fail.

In the process of game development the meaning and structure is expressed in the content the game designer and level designer provides. The game designer provides the development team with the overal picture of how the game should function. "Level designers 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." [8]. However, this job is not an easy one because for some "Level design is an art..." [8]. We believe that a helpful tool like PCG should be configured in such a way that the meaning and theme of the game and of its content is conveyed through it. The content in a game is not just randomized pieces but rather carefully placed pieces in carefully chosen positions. If a platform is unreachable in a platform game is has to have a purpose other than being there for the player character to walk on. It is not hard to imagine the frustration of a player trying to reach a reward that is out of reach. Level design is perhaps a form of art because of the balance between providing challenges and rewards. A challenge that is too simple or too makes the player bored or frustrated. The essence is to provide the content in a way that a player can rise up to challenges and barely make them. In state-of-the-art game development this is solved in the painstaiking process of playtesting and iterative design [8].It would therefore be interesting to try to utlize previous playtesting efforts and a possible way of doing this is to be able to read a previous level design and generate new content from this in such a way that the meaning of the content is not lost but still new.

1.3 A little less randomization, a little more variation, please

In a previously proposed taxonomy for PCG [24], the question of the "right amount of variation" in relation to different runs of an algorithm. This poses two questions concerning what the "right amount" of variation consists of, namely; 1) is the same amount of variation desired across all content types and 2) how much, and what type of, variation does the game designer want? It is very plausible that the game (or game mode) itself is in some way dictating how much the right amount of variation due to its design. Considering the job of the level designer there is a risk that the game becomes too varied so that the overall theme of the game gets lost or the meaning or structure of the game confuses the player. However, the aim is to create an enjoyable experience for the player. In the game industry this is achieved by carefully crafting the game and its content in relation to the play tests that are performed on invited players of the right target group. In PCG terms, randomization is necessary but not sufficient for automatic level design purposes, as the output of the generator must be controlled in order to fit with the meaning and structure of the game.

2. DESIGN PATTERNS

In the seventies, the architect Christopher Alexander developed a language of patterns. The intent was to allow individuals to express their ability to design with the aid of a informal grammar. "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core solution to that problem, in such a way that you can use the solution a million times over, without ever doing it the same way twice" [2]. In essence the pattern has two components (problem and solution) but the value lies not in the specific solution but in the generalization of a solution space. This way of thinking was brought to bear on computer software design in 1994, when Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides published a set of descriptions or solutions for how to solve common and recurring problems in object-oriented design. The solutions are not a finished design per se but instead templates that can be used in many different situations [9].

2.1 Design patterns in games

In games, design patterns can be seen as providing answers to problems faced by the game and/or level designer. However, we can also see each pattern as a problem posed by the designer to the player. If we view the content in a level as challenges or problems the player must find a solution in order to continue the progression through the game. The idea of automating the construction of problems with a given solution is a strategy to avoid the limitations of constraint checking, allowing content to be produced without creating impassable obstacles for the player. Furthermore, if we use previously play-tested problems we can with some certainty reuse problems that are fitting different skill sets and skill levels and thus provide more appropriate content for particular players or player types.

The seminal work of Björk and Holopainen introduced design patterns to game design, and provides the foundations for the contemporary discussion about the topic [3]. The book describes hundreds of design patterns, at different levels of abstraction and with reference to different game genres and tasks of game design. Here we will focus on patterns in the design of game levels (and similar spatial designs, e.g. maps and tracks) as opposed to e.g. patterns in game user interface design or rewards.

Hullett recently analysed levels of a common first-person shooter (FPS) game in order to find recurring design patterns that had an impact on gameplay [11]. The patterns he found include arenas, sniper positions and galleries, commonly seen in many FPS games.

The work that is most closely related to our current concern is the work already done on design patterns in platform games. Smith et al. [19] analysed the design of platform game levels, and later devised the Tanagra mixed-initiative level generator [20]. Tanagra uses a constraint solver to generate level geometry in interaction with a human designer. The geometry is generated according to a number of patterns. These patterns occur at two different levels, the single-beat (micro) level where patterns such as the "gap pattern" and "spring pattern" can be found, and at a slightly higher level, where patterns such as "valley" and "mesa" are composed of three micro-patterns each. These patterns are implemented with some flexibility, as the constraint solver can decide to stretch them to some extent to fit in the overall structure of the level.

Peter Mawhorter [13] describes a level generator for Super Mario Bros based on "occupancy-regulated extension" (ORE). The generator works by connecting a number of small level chunks like pieces of a puzzle, and generates levels of more novelty and interestingness than many comparable level generators. ORE could be seen as a compositional approach to implementing design patterns in procedural level generation. However, the description of ORE which can be found in the literature only allows for small and static (non-parametrised) level chunks.

3. COMBINING PCG AND DESIGN PATTERNS

PCG and design patterns could plausibly be combined in several different ways, even when limiting the context to level design. Perhaps the most straightforward way is creating compositional content generators, that view each pattern as a spatial design element and simply combine these elements by connecting them next to each other. This can be done either with static elements, as in Mawhorter's ORE, or with parameterized elements, as in Tanagra. Patterns could be connected sequentially in one dimension (as in Tanagra), two dimensions (as in ORE) or potentially in three dimensions. It is also conceivable to stack patterns, i.e. place several patterns at the same place. This would have the effect of modulating one pattern by another. Some patterns may fit well to certain patterns if it is placed before or after.

Another way of using patterns in the PCG process is to use patterns as objectives, e.g. as evaluation/fitness functions or constraints in search-based PCG. The existence of particular patterns could be seen as desirable or undesirable properties, biasing the search in content space so that the resulting content would be more likely to include or not include certain patterns. Such an approach could potentially lead to more variation than the composition-based approach, but is also more computationally expensive and harder to predict. An example of this approach is the "choke point" evaluation function in a recent attempt to evolve maps for the StarCraft real-time strategy game [23]. Maps which contain choke points are assigned higher fitness and the results of the level generator are therefore likely to contain this particular pattern.

With both approaches, patterns can be selected that are particularly well suited to a particular player, for example in order to maximise entertainment as predicted by a player model.

4. A PLUMBER IN A STRANGELY DESIGNED LAND

In the classic (action) platform game *SMB* [15] the player guides the protagonist Mario (in single player mode) through the world of the Mushroom kingdom where platforms, holes in the ground (gaps), huge green pipes, boxes and blocks acts as aid and obstacles. Furthermore the land seems to be filled with aggressive and deadly enemies like Goombas, Bloopers, Bullet Bills, Buzzy Beetles, Cheep-Cheeps, Hammer Bros., Koopa Troopas, Koopa Paratroopas, Lakitu, Piranha Plants, and Spinies.

SMB consists of 8 worlds with 4 levels each and 11 bonus areas. The bonus areas are often areas containing extra rewards like coins other ones contain warp zones. The warp zones functions as "portals" [19] to other worlds or levels other than the next in sequence allowing a more experienced player move through the game without risking the loss of Mario's "lives". The last level of each world (the levels named 1–4, 2–4, etc.) takes place inside castles where a fight against the main antagonist Bowser end each level. These "boss fight" levels are different than the other levels in such a way that contain long straight sections with few obstacles and end with a hanging bridge over a lava pit where Bowser is supposed to be dropped into.

In the following, we present a case study in finding level design patterns based on Super Mario Bros (*SMB*).

5. LOOKING FOR PATTERNS IN ALL THE RIGHT PLACES

In order to find patterns in *SMB* we apply a combination of heuristic analysis [6] and rhythm groups [5] [19]. Rhythm groups "are often fairly small, encapsulating challenging sections of gameplay" [19]. By dividing every level into sections separated by areas where no or limited threat is imposed upon the player's avatar we get reasonable sized sections to compare, group and classify into patterns.

Every level in SMB contain about 15 beats (only 10 beats in level 1–4 and the maximum 30 in 8–1 with an average of 15.5). However, not all of these beats are unique enough to qualify as patterns but *SMB* contains more than the 4 geometry patterns and the 4 multi-beat patterns similair to those used in Tanagra [20]. We base the names of the different groups on the names used in Tanagra [20]. It should be noted that we base our suggestions on the analysis of level 1–1, 1–2, 1–3, 2–1, 2–3, 3–1, 3–2, 3–3, 4–1, 4–2, 5–1, 5–2, 5–3, 6–1, 6–2, 7–1, 7–3, 8–1, 8–2 and 8–3. Thus ommiting 1–4, 2–4, 3–4, 4–4, 5–4, 6–4, 7–4 and 8–4 due to their focus on the fight with Bowser and 2–2 and 7–2 which have an underwater setting. The levels 4–3 and 6–3 includes a sort of timed platform which falls down if Mario remains too long on them which might yield a doubled number of suggested patterns due to the time limitation.

5.1 Examples of Super Mario Bros design patterns

In this section of the paper we intend to present our suggested patterns briefly together with illustrations of a few of those. The full list of discovered patterns can be seen in table 1. The illustrations aim to clarify our suggested patterns together with one or more solutions of how the player can solve the problem.

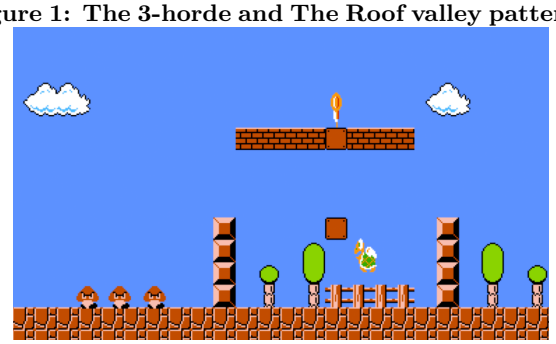
One could argue that the 23 suggested patterns are really only variations on five patterns, one for each group. This points to the problem of choosing a relevant level of abstraction when analysing a level into patterns. We have chosen a relatively fine-grained analysis, as we want to point out that there are meaningful differences in terms of gameplay between patterns that are superficially very similar. For example, two 2-hordes afford different solutions than one 4-horde (you could jump and land between the two enemy groups in the first case, but not in the second). We note that it would be plausible to see the five groups we identified as "macro-patterns" and the 23 patterns within them as "micro-patterns", but we will not pursue this semantic point any further here.

Figure 1 contains the two patterns "3-Horde" and "Roof valley". "3-Horde" can be solved with a triple short jump, a

Table 1: Patterns for Super Mario Bros. grouped by theme.

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 verticaly 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 differnt paths
3-Path	2 hanging platforms allowing Mario to choose differnt 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

Figure 1: The 3-horde and The Roof valley patterns.



long jump or a medium jump onto the first or last Goomba. The medium jump onto the last Goomba can with good timing allow the player to reach the pillar in the following "Roof valley"-pattern. The "Roof valley"-pattern is provided with a Koopa Paratroopa which, if timed properly, provides a boosted jump out of the valley. If the player misses the timing of the jump altogether, he is faced with a Paratroopa in the valley in which he has failed to jump out of.

Figure 2: The 3-horde and the Pillar gap patterns.

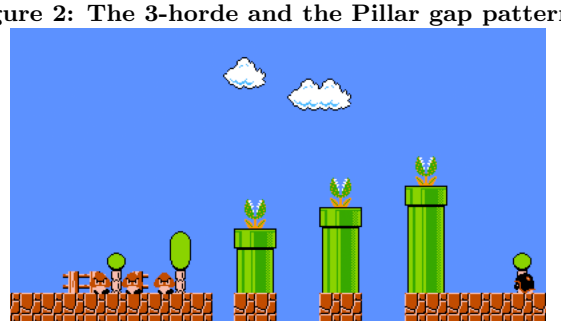


Figure 2 also contains two patterns; a now familiar "3-Horde" and a "Pillar gap"-pattern with Piranha plants forcing the player to both time the jumps with the movement of the plants in and out of the pipes movement as well as the escalating height of the pipes. By using Piranha plants a player may lose a life even though a correct "side-jump" between pipes normally would save him because of the plants moving in and out of the pipes. If a designer wished make this obstacle easier to pass he could remove the plants.

Figure 3: The Empty Valley and the Enemy Valley patterns.

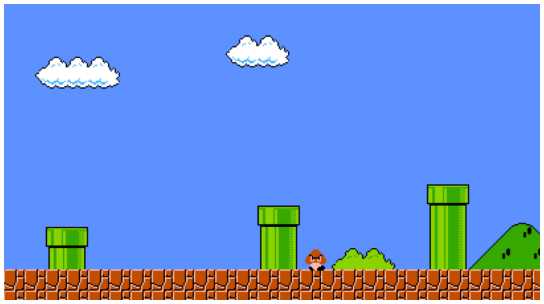
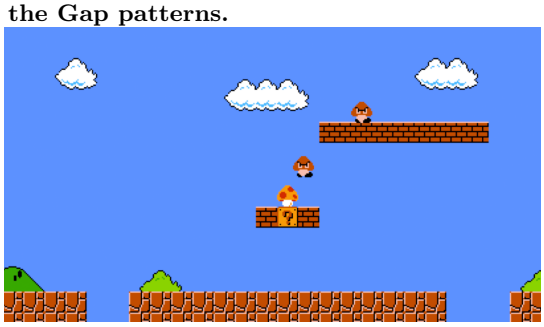


Figure 3 contains two patterns that share the same middle pipe. If we compare the first "Valley empty"-pattern with

the second "Valley enemy"-pattern we see that the difference in the height of the pipes suggest that the jump must be more precisely coordinated in relation to the pipe in the second pattern than in the first but if the player times the jump as a landing onto the Goomba, the bouncing of the Goomba will exert enough vertical speed that the ending pipe will be overshot.

Figure 4: The Gap,the 3-Path, the Risk and Reward and the Gap patterns.



In figure 4 we can see two simple "Gap"-patterns surrounding the more interesting combination of "2-Horde", "3-Path" and "Risk and reward" patterns. A power-up mushroom is placed in the second horizontal platform in such a way that two Goombas drops down upon the player if the timing is not right. If the player chooses to jump onto the question mark-block so that a Goomba is flipped the mushroom will go the other way and maybe leave the screen (in the original *SMB*, Mario can not go left to scroll the screen left) and the player does not get his reward.

5.1.1 In depth descriptions

Due to the limited space avaiable we have chosen to describe only a few of the patterns in more depth (see table 2, 3, 4, 5 and 6). We have chosen to exemplify the different groups of patterns by picking one from each group (see table 1).

6. PLAN FOR PATTERN-BASED MARIO LEVEL GENERATION

The previously suggested 23 patterns can be varied for enhancement of the play experience in several ways. For instance, the patterns with enemies can be varied by selecting different types of enemies. In *SMB* level 1–2 a Koopa Troopa is followed by two Goombas which enables the player to get variation in how to solve the situation; the player can either short jump and medium jump over the "Enemy" and "2-Horde" patterns or choose to jump onto the Koopa Troopa twice to use its shell to knock out the two Goombas. The prior solution is less risky than the latter and similairly the reward is lower with the prior solution. The player may also choose to run underneath the platforms and ignore the mushroom. The third possible path is approach the two Goombas a bit more cautiously and jump over them and then use the third platform to avoid the last gap altogether. Figure 5 may serve as an illustration on the previous situation replacing the Koopa Troopa with the two stand alone Goombas. By recognizing a player choosing the solution with the higher reward the PCG algorithm could generate patterns with higher demand on player skill. Within each

Table 2: 4-Horde Pattern Description.

Enemies – 4-Horde	
Problem	The player can act and traverse the level slowly without risk. The player masters the long jump and can therefore jump over multiple enemies.
Solution	By placing four enemies in a tight formation the maximum jump length is not enough to pass over the enemy which forces the player to use timing to land on any of the enemies for a second jump over the remaining enemies.
Using the pattern	Suitable use is on long platforms so that the enemies do not fall down any gaps. The pattern can also be used in conjunction with valleys to limit the landing area of the player. The pattern can be used to force the player to time running actions if they are placed on high platform allowing them to drop down on Mario.
Comments	Not all enemy types are suitable for this pattern. For instance, enemies that Mario cannot jump onto like Spiny may cause impassible sections of a level. Power-ups should always be considered when applying this pattern. If a too powerful power-up is placed in the wrong position the difficulty of the pattern can be drastically lowered. This pattern should not be misinterpreted as two 2-Horde patterns, the distance between the enemies in the formation is crucial.

Table 3: Pillar gap Pattern Description.

Gaps – Pillar gap	
Problem	The player can jump and traverse vertical obstacles without risk. The player masters high jumping and can therefore jump over high obstacles.
Solution	By placing a series of pillars with a limited width a less skilled player may overshoot a jump and miss the pillar and fall down the gap. The introduction of varying height of the pillars the player needs to master both vertical obstacles as well as the length of the jump.
Using the pattern	Suitable use is near the end of the level so that Mario is high enough to get to the top of the flagpole at the end of a level.
Comments	No power-up can save the player. But a skilled player may jump on the side of the pillar and perhaps bounce out of the gap danger.

Table 4: Enemy valley Pattern Description.

Valleys – Enemy valley	
Problem	The player can jump and traverse vertical obstacles without risk. The player can act and traverse the level slowly without risk.
Solution	By fencing in an enemy between two vertical obstacles they player is forced to engage the enemy without it falling through a gap. If a Koopa Troopa is used and the player jumps on it and then jumps on the shell the player will risk losing power-ups or a life due to the high-speeding shell bouncing between the vertical obstacles.
Using the pattern	The pattern can be used most types of enemies with the exclusion of Bullet Bill unless the distance between the vertical obstacles are placed with enough distance in between.
Comments	The pattern needs a sufficient platform length.

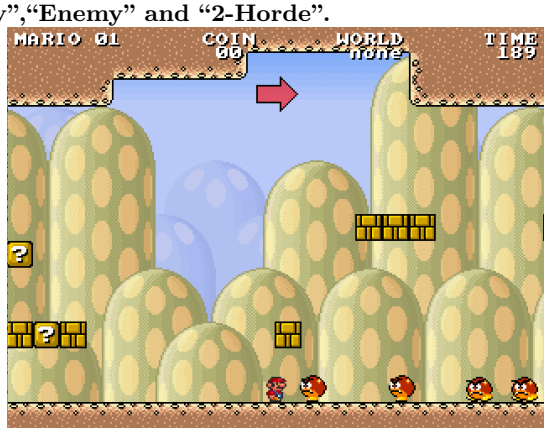
Table 5: Risk and Reward Pattern Description.

Multiple paths – Risk and Reward	
Problem	The level layout is more or less linear and the player's choice is limited.
Solution	Provide multiple paths where rewards, gaps and enemies are placed so that the player is forced to choose a specific way through this section of the level. The player can choose the specific path according to his/her skill and risk appetite as well as what the player consider their "favourite" obstacles or enemies to be. The need of a specific reward or power-up may also affect the choice.
Using the pattern	The primary use is to create variation. The pattern could also be used to introduce a needed reward or power-up or to add a more relaxed section of the level.
Comments	The introduction of a different type of decision making may affect the player's reaction time and therefore the distance between the beginning of the pattern and enemies and gaps must be thought through.

Table 6: Stair up Pattern Description.

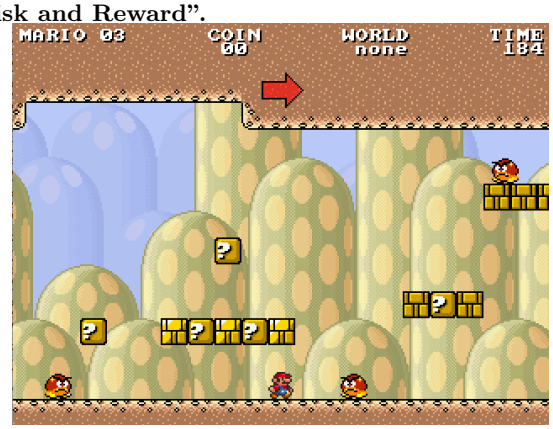
Stairs – Stair up	
Problem	The player needs to be on a different height and the player character cannot jump high enough.
Solution	By providing tightly placed platforms, blocks or pipes with increasing height the player can jump onto them and reach a higher position.
Using the pattern	The pattern is usable before any section where the player character needs to be high enough but unable to due to limitations in jump ability. In SMB it is often needed before the end of the level so that Mario may reach the highest point on the flagpole. It is also useful for variation before a multi-path pattern allowing the player to drop down instead of jumping up.
Comments	The stair should not be placed too high to reach for the player character or be so high that the player is limited in jumping by the top screen unless this is the intended effect.

Figure 5: Mario in a "Multiple path" facing "Enemy","Enemy" and "2-Horde".



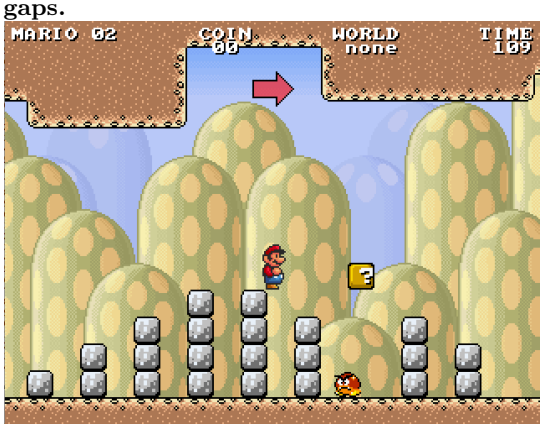
group of patterns a player that solves a pattern successfully can be faced with a pattern from the same group one step down the list. For all patterns the length of platforms can be changed both for variation and for difficulty. Similarly the patterns can with parametrization be varied in length and height as well as difficulty in conjunction with adding risk and rewards. Possible parameters are the number of gaps, the length of a gap, the length of a platform, enemy types, amount of enemies and if risk and reward should be present. Figure 6 illustrates a 3-Path and a Risk and Reward pattern combined. At the current stage, we have implemented

Figure 6: Mario leaving a "3-Path" and entering "Risk and Reward".



a composition-based level generator that randomly chooses among the 23 patterns, and generates playable levels for *Infinite Mario Bros*, (IMB), a public domain clone of Super Mario Bros that has been used extensively in game AI and PCG research [22]. It should be noted that IMB already contains a random level generator function with predefined sections. However, our level generator is based on existing content that was placed in SMB trying to take advantage of the effort that Nintendo put into designing and playtesting its original successful product. By doing so we hope that an experienced player will enjoy these levels as much as he did playing the original SMB but that the variation the randomization provides will keep the player interested for a longer time. In the immediate future, we will add functionality for parameterising the patterns, and selecting patterns to fit particular player profiles. We will also address the problem of preserving playability while stacking patterns (see figure 7); this will likely be done through simulation-based evaluation functions. We also intend to design metrics indicating when a player master a skill enough to be faced with a certain pattern and how often.

Figure 7: Mario in an interesting combination of pillars and "Stair-up", "Stair-down" and "Roof" without gaps.



7. CONCLUSION

In this short paper we have discussed the potential roles of design patterns in PCG, and presented an analysis of the levels in the original Super Mario Bros game into design patterns. Further, we discussed ways of creating levels in Super Mario Bros by combining these patterns. By ordering the patterns in sequence of difficulty we can vary the content in the new levels according to what a player does. In order to further vary the content we can use parametrization (platform length, gap length, enemy type, risk and reward) in conjunction with a specific pattern. The patterns can be placed in sequence or used together to create varied content.

8. ACKNOWLEDGMENTS

Thanks to the anonymous reviewers, Paul Davidsson, and Gillian Smith for insightful comments on the paper.

9. REFERENCES

[1] Acornsoft. Elite. [Digital game], 1984.

[2] C. Alexander, S. Ishikawa, and M. Silverstein. *A pattern language – towns, buildings, construction.* Oxford University Press, New York, U.S.A., 1977.

[3] S. Björk and J. Holopainen. *Patterns in game design.* Cengage Learning, 2005.

[4] Blizzard North. Diablo. [Digital game], 1996.

[5] K. Compton and M. Mateas. Procedural level design for platform games. In *Proceedings of the 2nd Artificial Intelligence and Interactive Digital Entertainment Conference*, 2006.

[6] 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.

[7] D. Dimovska, D. Wilson, K. Wong, L. Bojsen-Moeller, L. Korsgaard, M. Lyngvig, and R. D. Capua. Dark room sex game. [Digital game], 2008.

[8] T. Fullerton. *Game Design Workshop*. Morgan Kaufmann, New York, U.S.A., second edition, 2008.

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

[10] Gearbox Software. Borderlands. [Digital game], 2009.

[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*, pages 78–85, New York, NY, USA, 2010. ACM.

[12] Interactive Data Visualization, Inc. Speedtree. [Software], 2011.

[13] P. Mawhorter and M. Mateas. Procedural level generation using occupancy-regulated extension. In *Proceedings of the IEEE Conference on Computational Intelligence in Games (CIG)*, 2010.

[14] MicroProse. Civilization. [Digital game], 1991.

[15] Nintendo. Super mario bros. [Digital game], 1985.

[16] D. A. Norman. *The design of everyday things*. Basic Books, New York, U.S.A., 2002.

[17] Re-Logic. Terraria. [Digital game], 2011.

[18] K. Salen and E. Zimmerman. *Rules of Play*. MIT Press, Massachusetts, U.S.A., 2004.

[19] 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*, pages 75–80, New York, NY, USA, 2008. ACM.

[20] G. Smith, J. Whitehead, and M. Mateas. Tanagra: Reactive planning and constraint solving for mixed-initiative level design. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):201–215, 2011.

[21] F. Spufford. *Backroom Boys – The Secret Return of the British Boffin*. Faber and Faber Limited, Croydon, U.K., 2003.

[22] J. Togelius, S. Karakovskiy, and R. Baumgarten. The 2009 Mario AI Competition. In *CEC '10: Proceedings of the IEEE Congress on Evolutionary Computation*, 2010.

[23] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck, and G. N. Yannakakis. Multiobjective exploration of the starcraft map space. In *CIG '10: Proceedings of the IEEE Conference on Computational Intelligence and Games*, pages 265–272. IEEE, 2010.

[24] 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*, 3:172–186, 2011.