Exploring EPCG in The Witness

Nathan R. Sturtevant
Department of Computing Science
University of Alberta
Edmonton, AB, CANADA
nathanst@ualberta.ca

Abstract

This paper provides an examination of how Exhaustive Procedural Content Generation (EPCG) can be used to explore the creation of new content for games. We study the puzzles in the game The Witness along with an aesthetic for game design proposed by Jonathan Blow and Marc ten Bosch, showing how EPCG is well-suited to answering game design questions and for helping designers find unique content for their games.

Introduction

Procedural content generation (PCG) in games seems to have great potential. One could imagine a never-ending stream of deep and interesting games produced at little or no cost. But, games like No Man's Sky have shown that billions of combinations of content can still lead to repetitive and uninteresting play, and that it is easy for expectations to outstrip current technologies. That isn't to say that PCG will not one day deliver, but, similar to how updates to No Man's Sky have improved the game, more research is needed to better understand how to deliver compelling and interesting experiences.

One suggestion to improve PCG has been to involve a designer in the process, called mixed-initiative content creation (Liapis, Smith, and Shaker 2016). There are many ways that a computer can assist human designers, such as by ensuring that design constraints are enforced and that all areas in levels are reachable. This effort is primary supported by computer-aided design (CAD) tools.

Recent research (Sturtevant and Ota 2018) proposed a subfield of PCG called Exhaustive PCG, or EPCG. In this sub-area content is generated exhaustively or semi-exhaustively to fully explore a design space. In this paper we use a mixed-initiative form of EPCG to explore and answer questions about game design. By answering questions about all available content, an EPCG system is able to suggest meaningful content that can be selected for use in a game.[1] For instance, we might ask to see all puzzles of a certain size that have a single solution.

While in traditional PCG it is important to evaluate the output of content generators (Shaker, Smith, and Yannakakis 2016), in EPCG all content is potentially generated. Thus, it is also important to focus on the questions that can be asked about content, as these questions will determine the breadth of content that is generated.

We use a design aesthetic from Jonathan Blow and Marc ten Bosch to help focus the questions we ask about the available content.

A Matching Aesthetic

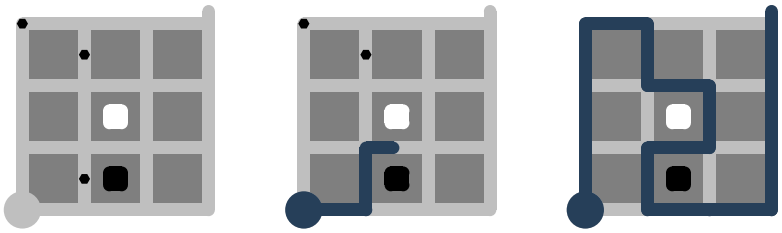
In a 2011 IndieCade presentation,[2] Jonathan Blow and Marc ten Bosch presented an aesthetic for game design which aims to, in their words, "show a lot of truth, with minimum contrivance." They suggest that in this aesthetic the role of a designer is to explore and reveal the truths about the universe more than to create a fun gameplay experience. They outlined eight pieces of this aesthetic as follows:

1. Richness
2. Completeness
3. Surprise
4. Lightest Contrivance
5. Strength of Boundary
6. Compatibility of Mechanics
7. Orthogonality
8. Generosity

The goal is to (1) aim for the richest space of game design, where there is significant room to explore the space of possible interactions between mechanics. Once that space is identified, the designer must (2) completely explore the interactions within the game before (3) editing and pruning the interactions to be left with a set that are interesting and surprising for the players. This aesthetic prefers mechanics that are (4) simple (with unnecessary complications) and (7) orthogonal (without overlapping functionality). At a broader level, the total set of mechanics explored should not overlap or be unbalanced, forming an interesting boundary between

[1] Taken broadly, EPCG includes some generic constraint programming or answer-set programming solvers, as such solvers may perform an exhaustive search with pruning (making them semi-exhaustive) over variables and values. However, these generic languages do not easily support many types of constraints that are used in games, such as the region constraints we discuss in this paper. In these situations, a custom solver may be more efficient and flexible.

[2] http://the-witness.net/news/2011/11/designing-to-reveal-the-nature-of-the-universe/



(a) (b) (c)

Figure 1: A sample puzzle with two types of constraints (a), the natural solution a player might try first (b), and the solution to the puzzle (c).

different puzzles (5), and each new mechanic should interact cleanly with existing mechanics (6). For the remainder of the paper we refer to the aesthetic as a whole as the BtB (Blow and ten Bosch) design aesthetic, and when referring specific aesthetic points, we will refer to them as (BtBN), where N is the aesthetic we are referring to.

In the BtB aesthetic, the goal isn't to explicitly make hard puzzles, but instead to look for truth about the design consequences in the space being explored and to illustrate it with a selection of puzzles, although the aesthetic is not limited to puzzles. This can be done in many ways; one such way would be to present a set of puzzles that follow a pattern and then to break that pattern to help the player to learn about the interactions of the mechanics.

EPCG broadly speaking is designed to exhaustively explore design spaces in ways that other more selective techniques do not. Thus, EPCG is well suited for the BtB aesthetic. For instance, due to its exhaustive nature, EPCG can explore the full consequences of different design decisions (BtB2). It can also explore how orthogonal constraints fit together (BtB6 and BtB7). But, EPCG does this best in a mixed-initiative process with a designer that is evaluating and editing content to maximize surprise and to minimize contrivance (BtB3 and BtB4).

Design in The Witness

In previous work we explored a single puzzle type from *The Witness* (Sturtevant and Ota 2018), a 2016 game by Jonathan Blow and Thekla Inc. In this paper we continue that exploration by looking at different puzzle constraints and how their mechanics can be put together using an EPCG approach.

To begin, we explain the basic mechanics of the puzzles in the game. A sample puzzle for game is shown in Figure 1, with the puzzle in part (a) and the solution in part (c). The goal of each puzzle is to trace a self-avoiding walk (Knuth 1976) from the start (the circle in the lower-level corner) to the goal (the notch in the upper-right corner) that obeys the constraints of the puzzle. Symbols are placed on the puzzle to indicate constraints on the path. This puzzle contains both types of constraints we study in this paper. The small hexagons are 'must cross' constraints, which indicate that the final solution path for the puzzle must cross that point. Hexagons can be placed on vertices or edges.

Table 1: Number of paths through a grid

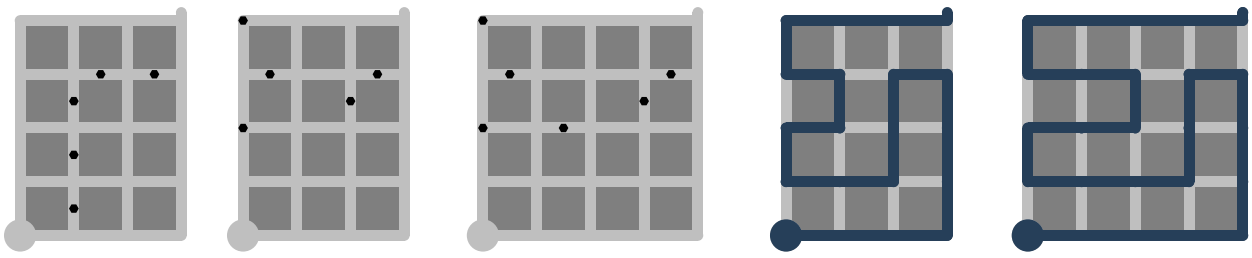
height	width	count
2	2	12
2	3	38
3	3	184
3	4	976
4	4	8,512
4	5	79,384
5	5	1,262,816

The large rounded squares are spatial constraints. In the final solution, the puzzle is divided into one or more regions by the solution path. No two spatial constraints of different colors can be in the same region in the final puzzle. This particular puzzle has black and white spatial constraints which must be separated by the path. The solution path splits the puzzle into two parts, with each part having one of the two spatial constraints.

This puzzle is interesting because visually the spatial constraints are grouped with the lowest must-cross constraint. Additionally, the spatial constraints are larger than the must-cross constraints, so one might focus on finding a path that satisfies these first. Thus, a natural first path to follow is what is shown in Figure 1(b). But, this path does not lead to a solution. So, this puzzle subverts the natural inclination of someone who is used to solving such puzzles and thus contains surprise (BtB3).

The number of self avoiding paths for a given table size is shown in Table 1. This grows exponentially, but there are efficient algorithms for computing the number of possible paths. The exponential growth is useful, because it eliminates the possibility of someone using brute-force to find a solution and suggests that there might be a rich space of solutions to explore (BtB1).

We employ a few general strategies when searching for interesting puzzles that lend themselves to the BtB aesthetic. First, since we are using EPCG, we fundamentally get a measure of completeness (BtB2). Within each portion of the space we explore, our search will be exhaustive. Second, related to using the lightest contrivance (BtB4), we will look for puzzles with as few constraints as possible. Placing more constraints on the board than is needed or using



(a) (b) (c) (d) (e)

Figure 2: Sample 3×4 and 4×4 puzzles. The solution to (b) is in (d), and the solution to (c) is in (e).

a larger board than is needed adds complexity but, for the puzzles analyzed here, does not significantly increase the quality of the puzzles. Finally, we prefer puzzles with fewer solutions, preferably only one. With fewer solutions players must demonstrate that they exactly understand the constraints given, which seems to give more opportunities for surprise (BtB3), although this may be specific to the particular puzzles studied here.

Must-Cross Constraints

To begin, we explore the must-cross constraints. First, how many constraints are there? In a puzzle width w and height h, where the puzzle in Figure 1 is 3×3, there are $w \times (h + 1) + (w + 1) \times h$ must-cross constraints that can be placed on edges (the right two constraints). An additional $(w+1) \times (h + 1)$ can be placed on vertices (the upper left constraint). Thus, the total number of positions where a constraint can be placed is $3wh+2w+2h+1$. If we then want to place k must-cross constraints onto a board with n possible constraints, there are $\binom{n}{k}$ ways to do so.

Our procedure for finding interesting puzzles works as follows. For each possible arrangement of constraints, we count how many legal solutions there are for a given puzzle. We then keep all puzzles with the minimum number of solutions. Initially we stored all such paths, but later we switched to only storing the longest paths. Longer paths may seem more contrived, but that piece of the aesthetic is pointing more towards mechanics, and we found that the longer paths tend to contain elements of surprise (BtB3) not found in shorter paths. For instance, consider the puzzle in Figure 2(a). Here the constraints are uninteresting as they only allow a single simple path to the goal.

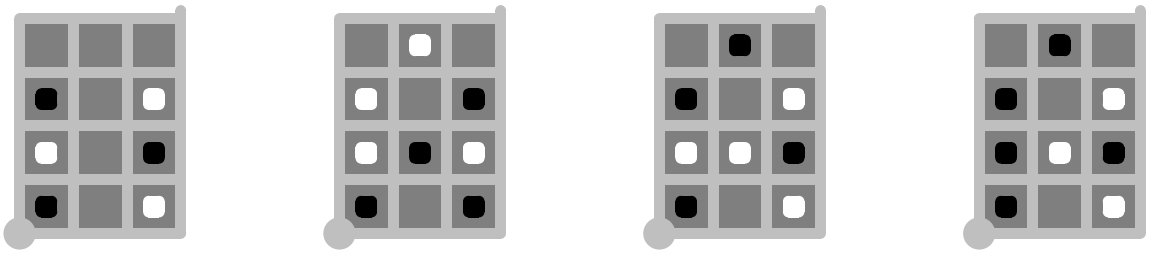
In the search process we gain some efficiency by throwing out a puzzle once we know that it is worse than the best puzzle found thus far. While significantly better performance could be achieved, our simple code was sufficient for our purposes – we able to explore all 4×4 boards with up to 6 constraints on each board in less than 1000 seconds, as shown in Table 2. In this table, the *max* column is the number of locations where constraints can be placed. The # column is the number of ways to place constraints. Note that for each possible placement, all combinations in Table 1 must be considered to compute the number of valid paths. That is, on a 3×3 board with 3 constraints we look at 9,880 possible puzzles. For each possible puzzle we look at 184 actual paths. So in the code we analyze $9,880 \cdot 184 = 1,817,920$ puzzle and path combinations together on a 3×3 board with 3 constraints. The *min* column is the minimum number of unique solutions for any puzzle that is generated, and the *time* column is the total time required to generate all puzzles and count the number of solutions. For the work in this paper, the time required to implement and test more efficient code would be larger than the time spent finding the solutions.

Our exploration revealed that 3×3 puzzles were too simple to be interesting. On the 3×4 board we found a few interesting puzzles, and these were not improved on significantly when moving to the 4×4 board. In Figure 2(b) we show one interesting puzzle with the solution in (d). The puzzle is interesting because the natural tendency is to draw a path directly towards the constraints, but the solution initially winds around the outside. When solving for 4×4 puzzles with 5 constraints on the board, there were no puzzles with unique solutions. With 6 constraints on the board we found the smaller puzzle from Figure 2(b) repeated with a single additional constraint, shown in Figure 2(c). As this does not bring significant new depth to the understanding of this puzzle and solution lengths were already maximal for the board size, we did not explore further, although for completeness it would be worthwhile to study puzzles with 7 constraints.

If we were to continue to explore these puzzles we would consider what happens when the board is filled with must-cross constraints with very few open locations. (The com-

Table 2: Number of must-cross constraints

height	width	max	#	total	min sol.	time
3	3	40	3	9,880	4	0.01s
3	3	40	5	658,008	1	1.51s
3	4	51	3	20,825	16	0.20s
3	4	51	5	2,349,060	1	5.42s
4	4	65	3	43,680	190	0.59s
4	4	65	4	677,040	27	6.01s
4	4	65	5	8,259,888	2	73.41s
4	4	65	6	82,598,880	1	969.74s



(a) (b) (c) (d)

Figure 3: One interesting set of separability constraint puzzles

plexity of putting must-cross constraints in all but k locations is the same as placing them in k locations.) In this case the constraints at the intersections of paths should be primarily used, as they give more options for movement. We could also explore the use of cannot-cross constraints, which are just broken lines in the game that cannot be crossed. There are several optional puzzles in *The Witness* that explore these together. Beyond this, there does not seem to be significant additional depth that can be found in must-cross constraints alone.

Region Constraints

As a second piece of exploration we look at the region constraints in a similar manner to our study of must-cross constraints. In a $w \times h$ puzzle there are $\binom{wh}{k}2^{k-1}$ ways to place k pieces onto the board, assuming two colors. For symmetry reduction the first piece placed is always black, and the remaining pieces can be any color.

As before, to get a sense of the size of the problem we show the number of combinations and minimum number of paths for solving a puzzle in Table 3. Although there are fewer puzzles for each board size, it is more expensive to compute whether a solution is valid because it involves dividing the board into regions; pre-computing the regions would speed up this calculation.

One interesting set of puzzles were found on the 3×4 board size, shown in Figure 3. The puzzle in (a) with 6 pieces has a natural beauty because of the placement of the pieces and their colors. Puzzles (b)-(d) have pieces in the same location in each puzzle, but the change in colors means that the solution used to solve each puzzle is completely different. (The solutions are in Figure 5 in the appendix.)

When describing the BtB aesthetic, Blow uses puzzles with the region constraints to illustrate several design points in the puzzles. The one point of completeness that is described and used in the game that we do not explore here is varying the location of the exit. Blow keeps the puzzles almost fixed and varies the exit to produce different paths; we do this with different colored constraints while keeping the constraint locations fixed. It would not be difficult to add start/goal locations as part of the EPCG exploration procedure.

For this paper we did not deeply explore more than two colors or the use of even larger boards. Most puzzles generated with three colors can be simplified to use only two colors, so further filtering is needed to find the interesting three-color puzzles. We have seen larger user-generated puzzles with just region constraints that are difficult to solve, so there is still room to explore region constraints on their own. One way to quickly reduce computational times on much larger boards is to only build puzzles with symmetric piece placement.

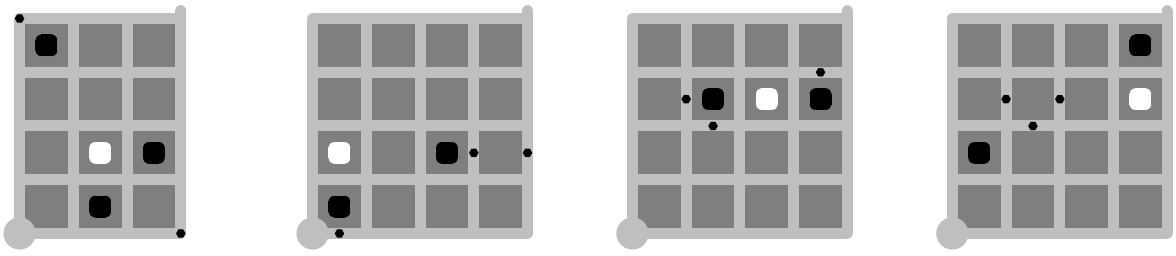
Table 3: Number of region constraints

height	width	max	#	total	min	time
3	3	9	3	336	14	0.01s
3	3	9	5	2,016	1	0.08s
3	4	12	3	880	87	0.24s
3	4	12	4	7920	20	0.59s
3	4	12	5	12,672	2	1.38s
4	4	16	3	4,480	820	6.25s
4	4	16	4	29,120	218	24.56s
4	4	16	5	69,888	74	74.36s
4	4	16	6	256,256	2	219.36s

Joint Must-Cross and Region Constraints

One point of the BtB aesthetic is that mechanics in games should have orthogonal design (BTB7). In the case of the constraints we are looking at in this paper, the constraints are physically placed in different locations on the puzzle - must-cross constraints are on paths, while region constraints are in the spaces between paths. This means that both types of constraints can be placed in the same puzzle, effectively combining the mechanics together. We previously discovered that small puzzles require too many constraints to produce puzzles with a single solution, and when a single solution is produced the solution tends to be trivial. Here we look at the impact of joining together the two mechanics. We find more interesting puzzles with the combination of constraints; it is an open question whether an interesting phase-transition occurs as the number of types of constraints is added to a puzzle (Gent and Walsh 1994).

Even adding a single must-cross constraint to a region constrained puzzle can make a puzzle significantly more



(a) (b) (c) (d)

Figure 4: Puzzles with joint must-cross and region constraints

complicated. Our previous work (Sturtevant and Ota 2018) studied how multiple puzzles can be combined together to create a *triple* puzzle where one solution solves all three sub-puzzles. That work did not take advantage of the orthogonality of constraints to use only a single puzzle with different types of constraints. But, it did suggest that when combining puzzles, it is useful to look for individual puzzles with the maximum number of solutions that, when combined together, only have a single solution. Although we did not use this metric directly when building puzzles, we were able to verify that it held for many puzzles.

There are 10,098,000 combinations of ways to place 2 must-cross and 4 region constraints on the 3×4 board. This takes 173.04s and leads to 485 unique puzzles (with maximum length and a single solution – there are 945 puzzles when including the shorter solutions). We illustrate one such puzzle in Figure 4(a). As with previous puzzles, this one is interesting because the natural first actions do not lead towards the solution.

On a 4x4 board there are 195,686,400 combinations of ways to place 3 must-cross and 3 region constraints on a 4×4 board. This takes 5721.52s and leads to 72 puzzles (with maximum length and two solutions each – there are 3020 puzzles when including the shorter solutions). We selected a few of these puzzles and placed them in Figure 4(b)-(d). All of these puzzles would be trivial to solve with just the region constraints, because there are so few region constraints. Similarly, they would be trivial to solve with just the must-cross constraints. But, when combined together, the resulting puzzles are non-trivial to solve. There are often many puzzles with the same solution but slightly different configurations of constraints. Part of the editing process that is needed after exploring puzzles is removing puzzles where the combinations of constraints do not seem to work well together. The puzzle in Figure 4(c) is a strong puzzle because the must-cross constraints interact closely with the region constraints. But, the puzzle in in Figure 4(d) is poor because the must-cross constraints are placed independently from the region constraints, making it feel like two separate puzzles instead of a single integrated puzzle. (The solution can also be pieced together more independently.) Because both of these puzzles have the same solutions, we would only want one of them in practice.

Given further exploration, it would be worth deeply exploring larger puzzles with as few constraints as possible to see if the use of two types of constraints is able to reduce the total number of constraints required to make interesting problems. It is also worth exploring together some of the many other constraints available in the game to see what new and interesting things can be learned through the EPCG process.

As a preliminary test of the value of these puzzles we uploaded the puzzles in Figure 4(a) and (b) to a popular site for *The Witness* puzzles[3]. The site keeps track of the most popular puzzles by 'likes', and within two days Figure 4(b) was tied for the most popular puzzle of the previous two weeks and Figure 4(a) was just one 'like' shy of Figure 4(b). While further user studies are needed, this provides preliminary evidence that users find these puzzles interesting to solve.

Summary

We would summarize the process we describe in this paper as one of exploring the nature of the constraints in *The Witness* to see what puzzles arise from these constraints. This is a mixed-initiative process, with a designer asking questions about the size of the board, the number of solutions, etc, and then analyzing the results to look for interesting content and new questions to ask.

From this process we make a few generalizations about puzzle design. First, puzzles with many solutions are less interesting because they are less likely to test specific understanding. Thus, a common EPCG question seems to be to ask for puzzles with a single solution, or with as few solutions as possible. Second, puzzles with fewer constraints seem to be more interesting than those with more constraints. Fewer constraints seem to reduce the contrivance of the puzzles and ensure that the puzzles do not overwhelm the reasoning capacity of the player solving the puzzles. Instead they directly test the understanding of the constraints in the game. Thus, a common EPCG process will likely involve iterating over small board with small numbers of constraints to find the threshold where interesting problems emerge. This is beneficial for EPCG because of its limited ability to scale to very large problem instances. Finally, we

[3]https://windmill.thefifthmatt.com

observe that, in this domain, avoiding very short paths inherently produces puzzles that have interesting interactions, even without directly optimizing for these interactions. In many of the puzzles shown in this paper there is a simple solution that can be achieved if a single constraint is removed. This minimalism pushes the player directly up against the constraints in each puzzle, forcing the player to analyze puzzle in new ways.

The work in this paper relied on writing custom code to answer the EPCG questions that arose. Future work will need to look at more accessible ways of using EPCG to explore game design that do not require as much code. This will allow more users to engage with EPCG methods and use them to explore their own design spaces.

References

Gent, I. P., and Walsh, T. 1994. The SAT phase transition. In *ECAI*, 105–109.

Knuth, D. E. 1976. Mathematics and computer science: Coping with finiteness. *Science* 194(4271):1235–1242.

Liapis, A.; Smith, G.; and Shaker, N. 2016. Mixed-initiative content creation. In Shaker, N.; Togelius, J.; and Nelson, M. J., eds., *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. 195–214.

Shaker, N.; Smith, G.; and Yannakakis, G. N. 2016. Evaluating content generators. In Shaker, N.; Togelius, J.; and Nelson, M. J., eds., *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Springer. 215–224.

Sturtevant, N. R., and Ota, M. J. 2018. Algorithms for exhaustive and semi-exhaustive procedural content generation. In *Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*.

Appendix: Puzzle Solutions

For completeness, in this section we provide the solutions to any puzzles in the paper that did not already appear.

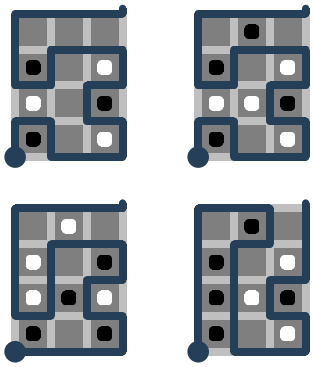


Figure 5: Solutions to puzzles in Figure 3

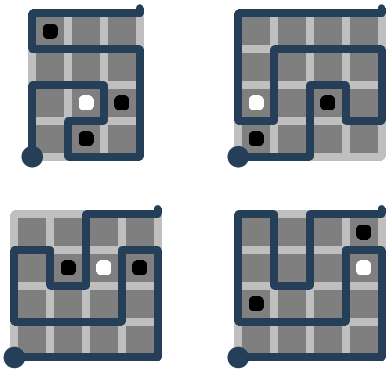


Figure 6: Solutions to puzzles in Figure 4