Combination: Automated Generation of Puzzles with Constraints

Christopher Jefferson
University of St. Andrews
caj@cs.st-andrews.ac.uk

Wendy Moncur
University of Aberdeen
wmoncur@abdn.ac.uk

Karen E. Petrie
University of Dundee
karenpetrie@computing.dundee.ac.uk

ABSTRACT

Constraint Programming offers a powerful means of solving a wide variety of combinatorial problems. We have used this powerful paradigm to create a successful computer game called Combination. Combination is an application for the iPhone and iPod touch. It has been on sale internationally through the iTunes store[1] since December, 2008 and received a number of positive reviews.

In this paper we explain how all the levels of Combination were generated, checked for correctness and rated for difficulty completely automatically through the use of constraints. We go on to evaluate this method of creation with the use of a human evaluation. This showed that fun, immersing computer games can be created with constraint programming.

1. INTRODUCTION

Combination is a puzzle game, loosely related to chess-based problems. The premise of the puzzle is to place wooden pieces that emit colour rays, into a grid so that no coloured ray hits a piece of another colour. There are three types of squares in the grid:

- Empty Squares, where pieces can be placed;
- Walls, which cannot contain pieces and block rays;
- Holes, which cannot contain pieces but do not block rays.

Each piece has a colour. A piece emits rays of this colour, and must not be hit by a ray of any other colour. Small balls on each of the pieces show both the colour of the piece and which direction the coloured rays will be emitted from. There are three different possible piece configurations:

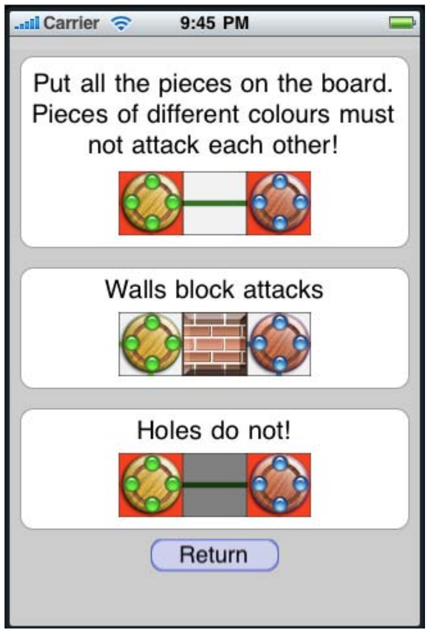
- Pieces that emit rays from the top, bottom, left and right;
- Pieces that emit rays from all 4 of the diagonals;
- Pieces that combine both of the above pieces to emit rays from 8 directions.

[1]The game can be seen or downloaded from: http://itunes.apple.com/app/combination/id300288142?mt=8

SAC'11, 21-MAR-2011, TaiChung, Taiwan Copyright ©2010 ACM 978-1-4503-0113-8/11/03. . . $10.00

A level is completed when all the pieces are placed on the grid and no piece is hit by a ray of a different colour. Figure 1 shows the instructions that players of the game are given. Figure 2 shows an easy level of the game. Figure 3 shows the solution to a more complex level.

Rays are blocked both by walls, and by other pieces. For example in Figure 3 the top right-hand piece would attack the pieces in the bottom right, but is blocked by the piece immediately below it, of the same colour.



Put all the pieces on the board. Pieces of different colours must not attack each other!

Walls block attacks

Holes do not!

Figure 1: Instructions screen

Combination uses AI in two ways, fully automating the process of generating the content of the game. Firstly, the levels are all generated automatically using a combination of *Constraint Programming* (CP) and local search. Secondly, the difficulty of each level is graded using CP.

In the next section we discuss the past work in solving puzzles with CP. In the subsequent section we outline the CP model used and the solving techniques explored. The subsequent section contains a brief human based evaluation of the game, with an empirical evaluation. The paper concludes by looking back at the mistakes we made when creating the game and by outlining how successful the game has been commercially.

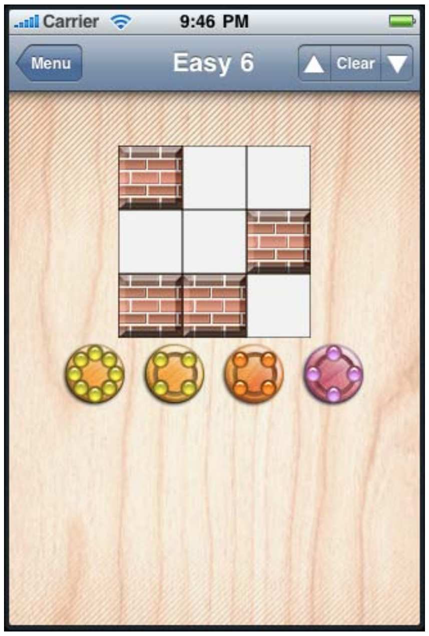


Figure 2: A level from early in the game

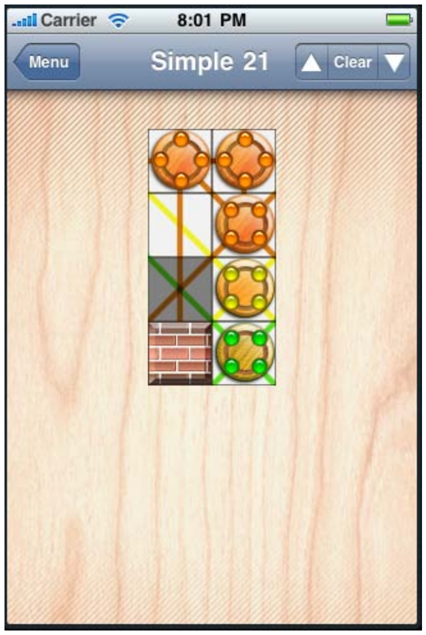


Figure 3: Example of a completed level with Blocking

2. BACKGROUND

A constraint satisfaction problem is a triple $(\mathcal{X}, D, \mathcal{C})$, where $\mathcal{X}$ is a finite set of variables. For each variable $x \in \mathcal{X}$ there is a finite set of values $D(x)$ (its *domain*). The finite set $\mathcal{C}$ consists of constraints on the variables. Each constraint $c \in \mathcal{C}$ is defined over a sequence, $\mathcal{X}'$, of variables in $\mathcal{X}$. A subset of the Cartesian product of the domains of the members of $\mathcal{X}'$ gives the set of allowed combinations of values. A *complete assignment* maps every variable to a member of its domain. A complete assignment that satisfies all constraints is a *solution*.

Combination is loosely based upon the puzzle Peaceably Coexisting Armies of Queens, levels of which have been solved optimally by CP [13]. In the 'Armies of Queen' problem you are required to place two equally-sized armies of black and white queens on a chessboard so that the white queens do not attack the black queens (and necessarily vice versa). In Combination you can have more than two colours of armies, represented by the different coloured balls on the pieces. You can also have both rooks and bishops (in chess terms) as well as queens. There is also no requirement that the pieces must be in equally sized armies. In addition, we added the concept of holes and walls to the grid as it leads to more interesting small puzzles. Due to all these changes, there was very little we could take from the original paper in terms of CP.

Solving puzzles has long been a favourite pastime for the constraint programming community. Back in 1978 Lauriere [6] used puzzles to come up with new constraint mechanisms and to compare different solution methods. Puzzles that have been studied include n-queens [4], Sudoku [10], Karuko [11] and Shikaku [12].

The main usage of CP to measure difficulty in the past has been with pure-reasoning games. For example, when solving Sudoku puzzles, it is generally assumed players should not try out values and then rub them out later. This means that with a sufficiently good set of reasoning rules, all Sudoku should be solvable without search. Much of the work in CP has been to try to improve reasoning, to avoid search [10].

Many other puzzles, including Combination, do not fall into this category. It is not intended that users should be able to solve games without search or trial. Other games in this category include 'Rush Hour'[2] and 'Subway Shuffle'[3]. Practitioners have successfully solved levels of such puzzles with CP, such as the Peacebly Coexisting Armies of Queens puzzle mentioned earlier and Alien Tiles [3]. However, we are unaware of any published literature explaining how these puzzles have been designed and the levels graded with the use of CP.

3. MODELLING COMBINATION

An instance of Combination is a quadruple $\langle x, y, G, P\rangle$. G is a 2D grid with dimensions $\langle x, y\rangle$, where each grid location is labelled by one of $\{wall, hole, empty\}$. P is a list of triples of length L_P of the form $\langle count, col, attack\rangle$, which denote that there are *count* pieces of colour *col* which attack in the directions given by $attack$. We allow each member of P to have a *count* for efficiency, the model works correctly if multiple elements of P have the same *col* and $attack$.

We model this as a CSP with a single 3D matrix of Booleans $M[x, y, L_P]$. $M[i, j, k]$ is true if there is a piece on the kth type at grid location i, j. The constraints on M are:

1. The correct number of each type of piece must appear: $(\sum_{i=1}^{x}\sum_{j=1}^{y} M[i, j, k]) = c_k$, where c_k is the *count* of $P[k]$.
2. There can be no piece where there is a hole or a wall: $\sum_{i=1}^{k} M[i, j] = 0$ if $G(i, j) \in \{hole, wall\}$.

Finally, there are the constraints which ensure that no two pieces of different colours can attack each other. These constraints are generated specially for each grid and set of pieces, according to the following rules:

For every $k_1, k_2 \in \{1, \ldots, L_P\}$ where the *col* of $P[k_1]$ and $P[k_2]$ are different, and grid locations $L_1 = \langle i_1, j_1\rangle$, $L_2 = \langle i_2, j_2\rangle$, such that piece k_1 can attack location L_2 from L_1 (which implies both it can attack in that direction, and there are no wall pieces

[2]http://www.thinkfun.com/rushhour/

[3]http://www.subwayshuffle.com/

along the path), then add the following constraint, implemented in Minion using the WatchedOr constraint:

$$M[i_1, j_1, k_1] = 0 \vee M[i_2, j_2, k_2] = 0 \vee \\ \exists (i, j) \text{ on the path between } L_1 \text{ and } L_2. \\ k \in \{1, \dots, L_P\}. \ (M[i, j, k] \neq 0)$$

This constraint can be read as saying that either there is no piece of type k_1 at L_1 or no piece of type k_2 at L_2, or there is some piece on the path between L_1 and L_2. It is not necessary to check if a piece between L_1 and L_2 is attacked by a piece of type k_1 or L_1 in this constraint, as separate constraints will deal with the relation between this piece, and the pieces at L_1 and L_2. Note that this blocking means it is not the case that an unsolvable Combination problem remains unsolvable if extra pieces are added.

We also add simple static symmetry breaking to our model, using the Crawford order [9]. We considered two types of symmetry. Firstly we consider colour interchangability. This is where there are two colours which have identical numbers of pieces with each kind of attack. Such a symmetry will generate a permutation of P which maps one symmetric colour to the other, which can in turn be mapped to a permutation of the matrix M, which maps solutions to solutions. This is used to generate Crawford ordering constraints as described in [9].

There can also be symmetries of the grid G, which can also be turned into symmetries of M. We do not use these symmetries, because we found from experience that users expected problems having one solution to include breaking the colour but not grid symmetry.

There are many more advanced features we could use to improve this model, this is by no means the most efficient CP model we could construct. However, we do not want to make our model too complex, else the search trees generated will be too small to accurately compare easier levels. In future we intend to investigate if some models better measure difficulty than others.

4. EVALUATING DIFFICULTY OF LEVELS

We hypothesise that constraint programming is similar to the reasoning which people do when solving problems. Therefore a good method of judging the relative difficulty of a problem should be to look at how hard it is for a CSP solver to solve. However, this method has one major limitation. Depending on the search ordering used, it is possible a solution may be found almost immediately, without any significant work.

In early betas of Combination, users noticed that on easy levels, they should try to first place the first few pieces into the top-left hand corner, as this was the search ordered used and so was the first place the solver tried when searching. Hence, the levels were not ordered well in terms of actual difficulty.

One alternative algorithm we tried was to use more advanced variable orderings. We could choose a variable ordering which had to make very many less choices based on an arbitrary ordering of the variables, by example looking at how many constraints are on each variable, and how tight these constraints are. Such variable orderings however still suffer from the problem that they occasionally find the solutions to very hard problems by luck, rather than by good design. Also, levels they designate as easy may require some very complex reasoning set very early on in the solution process.

To remove these biases, we used a new algorithm with solves each problem many times. This algorithm takes the average of the size of the search tree over many solving attempts, typically 50, each time randomising the initial ordering of the variables given to the solver. This removes any bias always placing pieces in the same area of the grid, and removes the chance of finding a solution early by luck.

	min	max	mean	standard deviation
Level Four	3	29	13.8	6.4
Level Six	4	123	36.1	23.3

Table 1: Distribution of search size for two problems

	min	max	mean	standard deviation
Level Four	11	15	13.3	0.88
Level Six	28	43	36.3	3.39

Table 2: Distribution of search size for two problems, averaged over 50 solver executions

As a concrete example we consider in depth the fourth and sixth hardest levels in the experiment in Section 6, using the model described in Section 3. We solved each of these instances 300 times, with different variable orderings. Table 1 provides a statistical analysis of the results.

If we take a single random sample from each of the two distributions in Table 1, Level Four takes fewer search nodes than Level Six 86% of the time. This clearly does not provide a reliable way of comparing these two problems.

Table 2 samples a different distribution, where each problem is solved 50 time and the average taken. These new distributions model our expectations, for we would expect averaging a normal distribution to reduce the standard deviation by a factor of $\sqrt{50} \approx 7$. As the table clearly shows, in this new averaged distribution Level Four always takes few nodes to solve than Level Six. Using this technique gives us a reliable way of measuring the comparative difficulty of levels. This does not of course mean this is a useful way of measuring how difficult a problem is for humans.

4.1 Reducing the solving time

For some hard problems, our basic model would take too long to solve. One of the most useful methods for reducing the size of search, and also reducing the search time for many instances, is Singleton Arc Consistency (SAC) [1]. SAC is a generic method of reducing search in propagating, branching constraint solvers.

We also investigated adding a selection of implied constraints to reduce search. By far the most useful implied constraint we found was the Global Cardinality Constraint (GCC) [8]. This constraint combines all the constraints in the Combination model of type (1), given in Section 3, which impose that the correct number of pieces of each type must appear.

We found that both of these improvements made substantial improvements to how quickly hard levels could be solved. For hard levels they made only very small changes to the ordering of difficulty of levels. For easier levels these techniques lead to many easy instances being solved in a very small number of nodes. In particular, similar to solving Sudoku with higher levels of reasoning [10], with both SAC and GCC, many easy levels were solvable without backtrack. This made it much harder to distinguish between these levels. We could in principle have tried to count how often constraints were triggered and how much work they did, but we preferred to avoid this complication.

In practice we introduced a cut-off, where both SAC and GCC were used first and then if the number of nodes taken to solve was less than 500, they were both turned off so a more precise complexity figure could be generated. This generated a good balance between accurately measuring the difficulty, while still allowing the most difficult problems to be solved in a reasonable time.

5. GENERATING COMBINATION LEVELS

Section 4 showed how we measure the difficulty of a level of Combination. In order to build a completely automated system we must also generate levels.

In this paper we do not generate levels of a specific difficulty level. Instead, we generate a large set of levels and measure the difficulty level of each. Generating each level and measuring its difficulty usually takes a fraction of a second, and less than ten seconds at worst. Therefore on a Dual Core 2.4Ghz laptop it is possible to generate tens of thousands of levels a day. We then generate a database of many thousands of levels and their difficulty level which can be used to either construct a single game, or provide players with a random level of a particular difficulty.

We first tried generating completely random instances of the Combination problem, but such instances were almost always either trivially solvable or trivially unsolvable. To generate interesting problems we instead use a local search strategy, which we combine with our CP model and method of measuring the difficulty of levels, discussed in Section 4.

At each step of the local search, if the level being considered has only one solution it is recorded in a database, along with its difficulty. As discussed in Section 3, based on user feedback we apply symmetry breaking to the pieces but not the grid when deciding if the problem has only one solution.

The local search technique we used generates a random grid and then adds pieces in turn, at each stage making sure the problem is still solvable. After 25 steps, the current grid is abandoned and the algorithm is restarted with a new random grid. 25 steps was chosen by experience. The exact value of this variable does not appear too important, as long as it is higher than the most pieces we expect to ever place on the grid.

In local search, it is usual to have a fitness function, and choose between various next moves by which is the most fit. We can bias our algorithm towards more difficult or easier problems by changing how we calculate the fitness function. It is however not possible to generate a particular difficulty of problem, because the difficulty of a problem can either grow or shrink significantly as more pieces are added. In general problems become steadily harder as more pieces are added, and then slightly easier when we reach the point where they have exactly one or no solutions.

There are cases where adding multiple pieces to a solvable Combination puzzle leads to a solvable problem, but adding the pieces one at a time leads to insolvable problems. We found by experiment that these are often the hardest instances. Therefore when searching for very hard instances, we sometimes add multiple pieces at the same time.

5.1 Level Post-Processing

While our basic algorithm produces correct levels and and measures the difficulty, over time we have added extra "prettyness" filtering rules, which are applied to a level to generate a more aesthetically pleasing level of identical difficulty. The two most important rules we apply are removing entire lines of walls and holes all along one edge of the grid, and changing a wall to a hole when it is impossible for it to block a piece, for example because it is in a corner.

6. EVALUATION

In Section 4 we outlined the method we used to evaluate the difficulty of levels. We conjectured that solving the CP once using a single ordering heuristic (hereafter refereed to as the *single method*) is not as good for our purposes as running the CP a number of times using different variable orderings then averaging the result (hereafter refereed to as the *multiple method*). In order to show our conjecture is correct we need to prove the following hypotheses:

H1: The multiple method evaluates the hardness for humans best.

H2: The multiple method will provide a more satisfactory user experience.

A between-subjects design was used to evaluate (i) the optimal algorithm for generating levels of appropriate difficulty, based on user performance, and (ii) user experience of the game. We recruited an opportunity sample of participants (n=18: 12 male, 6 female) via an online notice board at the local university. An incentive was offered of entry into a prize draw for an iPod Shuffle. All participants were able-bodied, and able to use an Apple iPod Touch without assistance. Participants were assigned at random to one of two groups. Each group tested versions of Combination which contained the same levels, but ordered using a different algorithm.

Group1: 6 levels evaluated for difficulty with the multiple method.

Group2: 6 levels evaluated for difficulty with the single method.

There were five stages to the study:
1. Participants gave their consent to take part after reading information about the study, and provided brief demographic information.
2. They undertook a brief training task, by completing a tutorial which first explained how to play the game, then gave them four practice games for them to try out.
3. Participants were then given 15 minutes to play up to six levels of the game. Each game level was intended to increase in difficulty compared to the last one. All participants began at level 1. For each level except the first one played, participants identified whether the level was harder or easier than the preceding one, using a seven-point Likert scale (1= Extremely hard to 7= Extremely easy). An eighth point on the scale existed (0 = Impossible). This was marked when participants failed to complete the level.
4. Participants then answered questions to establish the extent of their immersion in the game, using a subset of the validated questionnaire developed by Jennett et al [5]. Basic attention, temporal dissociation (losing track of time), challenge and enjoyment were evaluated using a 5-point Likert scale. Transportation (a sense of being removed to another place) and emotional involvement were not measured, as these were more relevant to online gaming than to a puzzle such as Combination.
5. There was an opportunity for participants to ask further questions about the study after completing the questionnaire.

6.1 Results

Results of the questionnaire were analysed quantitatively using independent samples t-tests. Participants found that the level of difficulty fluctuated between games, rather than increasing steadily. This fluctuation was more pronounced for those in Group2. There was a significant difference in the degree of difficulty between levels 1 and 2 for Group1 (M=5.10 , SE=0.348) and Group2 (M=4.33 , SE=0.726): t(17)=0.984, p=0.040. There was also a significant difference in the degree of difficulty between levels 5 and 6, comparing Group1 (M=1.40 , SE=0.542) and Group2 (M=2.33 , SE=1.167): t(17)=-0.751, p=0.002. This means that there was a bigger jump in difficulty between levels 1 to 2, and 5 to 6, for Group2 than there was for Group1.

The amount of time each participant spent completing each level is shown in Table 3. As these tests were all done by different participants they are rather difficult to analyse. However, what is obvious is that more people in Group1 completed more levels than those in Group2. We believe that this shows that the learning curve for Group1 was better than that for Group2.

In particular the average time to solve each level for Group1 varied between 56 seconds for level 2 to 2 minutes 44 seconds for level 3, whereas Group2 clearly found level 3 very difficult, taking an average of 6 minutes for those who could solve it, and only 18 seconds to solve level 6.

Given these results and results discussed earlier, we believe we have strong preliminary evidence to show Hypothesis 1 is true.

Overall, participants found that the game delivered an immersive experience, that captured their attention, provided a distraction, and was challenging and enjoyable as shown by the results in Table 4.

There was no significant difference between the degree of immersion reported by participants in the two groups, based on quantitative analysis of answers to the questionnaire. There was no significant difference for total immersion, basic attention, temporal dissociation (losing track of time) or challenge. However, the difference in enjoyment for Group1 (M=14.1, SE=1.21) and Group2 (M=14.56, SE=0.65) approached significance: t(17)=-0.320, p = 0.059. This result suggests that those in Group2 found that their algorithm generated slightly more enjoyable games. Therefore we have evidence that H2 is disproven.

6.2 Discussion

It is difficult to apply standard measures of usability – efficiency, effectiveness and user satisfaction – equally in the gaming domain. Efficiency can scarcely be regarded as a desirable goal in game playing, as the lack of challenges arising from efficiency reduces or eliminates user satisfaction [2]. We focussed specifically on effectiveness and user satisfaction, as we wanted to know if the game was fun to use and created a sense of immersion – the "specific, psychological experience of engaging with a computer game" [5]. The results show that the game is indeed fun to play and immersive. This is not influenced greatly by whether the multiple method or single method is used to evaluate level difficulty, even though the single method generates game levels that are erratic in their difficulty level. Users apparently enjoyed this erratic behaviour - as shown by the slightly higher enjoyment scores for the singe method. However, this was not borne out by observations of participants. Those in Group2 were notable for their audible expressions of frustration, and requests for hints on how to complete a level, addressed to the researcher.

The results so far show that the game provides a simple but effective immersive experience. It may be improved by adding a "Hint" feature, and a tutorial with more complex examples as both these feature were requested by participants. The choice of algorithm does not impact significantly on the quality of user experience. Caution is appropriate however. This study was formative. The number of participants failed to reach statistical significance. To confirm the results from this study, a larger field study would be needed. Ideally, we would also gather and analyse objective physiological measures such as heart rate [7]. However, this is beyond the scope of a small scale formative evaluation such as the one reported on here.

7. WHAT WENT WRONG?

Both based on our evaluation, and on discussions with users and beta-testers, constraint programming succeeded in the primary objective we set for it. All levels in the game were automatically generated, checked correct and organised in order of difficulty by constraint programming.

One common complaint is that for many users the game increases in difficulty too quickly, and the later levels are too hard for all but the very best puzzle solvers. One further, surprising result occurred when the number of levels in Combination were increased from around 50 to the current 393. Originally most good players would find the game begin to get very hard around level 40. They would however try hard to continue through the final 10 levels. Keeping the difficulty curve the same, the same position in the game now occurs when there are around 80 levels left. Informal discussions with players suggest they would see the end of the game as sufficiently far off that they abandon trying to finish.

The important conclusion to draw from this is that simply generating levels is insufficient for a complete game, it is important to investigate how users will interact with these levels and the game as a whole. In particular we noticed in our experiments that users appear to enjoy the difficultly level to increase and decrease. We do not believe this changes the usefulness of our algorithm, only how we should then present levels to the users.

8. COMMERCIAL SUCCESS

Combination was released on the iTunes store in December 2008[4], For £0.79, or $0.99. Despite no advertising of any kind the full game which features 397 levels has been sold 800 times. This has raised a total income of approximately £600. Given that the game was relatively easy for a constraint practitioner to model and hence produce levels for and that the entire game was completed in a week of evenings, we feel that this is good return. Based on our experiences with this game, we now think we could make further games with very little time overhead. Further, we believe investing in advertising could make our current game much more successful, given the reviews it has received.

There is also a free version of the game called Combination Lite which provides 16 of the levels from the full game. This has been downloaded 14,000 times. Many of the sales of the full version have been from people who have finished the lite version and wish to continue playing.

One of the feature of the iTunes store is the ability to review games. Internationally (it should be noted that each country has its own iTunes store) we have received 24 reviews, of which 22 gave the game the maximum 5 starts.

The game has also received one review in the media[5] which is very positive.

9. CONCLUSION AND FUTURE WORK

Combination is the first exploration of creating a computer game, in the puzzle game genre, with the use of CP. We used CP to automatically generate all the levels of Combination. We then checked all the games levels for correctness and rated their difficulty. Our human evaluation showed that our method of solving the problem multiple times with different ordering heuristics, is indeed a good measure of difficulty. Rather surprisingly, the evaluation of the questionnaire related to this evaluation also suggested that game players do not like the levels to be necessarily in increasing level of difficulty. This is something we were very surprised by, that may warrant further evaluation by the HCI community. The main point to note from the evaluation is that on the whole participants enjoyed playing the game. This enjoyment is also echoed in the reviews the

[4]http://itunes.apple.com/app/combination/id300288142?mt=8

[5]http://arstechnica.com/apple/reviews/2009/04/minireview-combination-offers-challenging-iphone-puzzles.ars

Participants in Group 1						
	level 1	level 2	level 3	level 4	level 5	level 6
Participant 1	0:02:55.541	0:00:38.046	0:00:35.517	0:00:27.549	0:02:01.209	-
Participant 2	0:04:04.252	0:00:50.850	0:03:12.927	0:00:35.932	0:00:55.144	0:00:51.500
Participant 3	0:00:44.379	0:00:45.719	0:00:25.737	0:00:23.197	0:01:37.006	0:00:35.350
Participant 4	0:02:26.649	0:00:24.288	0:03:30.722	0:00:33.237	-	-
Participant 5	0:00:18.047	0:01:08.640	0:06:55.012	0:05:38.024	-	-
Participant 6	0:00:19.550	0:00:42.143	0:00:39.119	0:01:29.465	0:03:55.178	0:01:25.072
Participant 7	0:00:15.837	0:00:31.190	0:00:28.459	0:00:28.498	0:01:29.077	0:01:20.822
Participant 8	0:02:50.754	0:03:17.112	0:08:55.390	-	-	-
Participant 9	0:01:56.817	0:00:38.218	0:01:51.395	0:01:23.691	-	-
Participant 10	-	0:00:27.240	0:00:50.064	0:01:45.261	0:01:29.980	-
Participants in Group 2						
	level 1	level 2	level 3	level 4	level 5	level 6
Participant 11	0:03:41.301	0:00:40.420	0:02:57.661	0:00:59.081	0:04:22.625	0:00:16.958
Participant 12	0:00:59.551	0:00:32.011	0:04:03.433	0:01:56.987	0:01:49.254	0:00:24.142
Participant 13	0:03:29.665	0:00:55.281	-	-	-	-
Participant 14	0:02:53.815	0:00:15.639	0:03:47.949	0:00:55.988	0:00:24.820	0:00:14.838
Participant 15	0:03:00.903	0:00:37.143	0:13:34.777	-	-	-
Participant 16	0:02:16.738	0:02:33.981	-	-	-	-
Participant 17	0:06:30.778	0:01:33.781	-	-	-	-
Participant 18	-	-	-	-	-	-

Table 3: The times taken for all of our participants to complete each of the levels they were given.

Category	Mean	Standard Deviation	Mean as percentage of max score	Max possible score
Total immersion	75.53	10.611	75.53	100
Basic attention	17.05	3.135	85.25	20
Temporal dissociation	21.22	3.770	70.73	30
Challenge	21.05	2.953	70.17	30
Enjoyment	14.32	3.019	71.60	20

Table 4: Analysis of how the Participants completed the Questionaire.

game has received. We did make mistakes when creating this game, in particular the later levels are possibly too difficult and there are too many off them. However, despite these mistakes, Combination has been a qualified commercial success.

in the future, we are going to use the lessons we have learned from this game to create further puzzle compilations. We would also like to explore the collaboration opportunities between HCI and CP.

10. REFERENCES

[1] C. Bessiere and R. Debruyne. Theoretical analysis of singleton arc consistency and its extensions. *Artif. Intell.*, 172:29–41, January 2008.

[2] M. Federoff. Heuristics and usability guidelines for the creation and evaluation of fun in video games. Unpublished Master of Science, Indiana University,, 2002.

[3] I. P. Gent, S. Linton, and B. M. Smith. Symmetry breaking in the alien tiles puzzle. Report APES-22-2000,, 2000.

[4] R. Haralik and G. Elliott. Increasing tree efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[5] C. Jennett, A. L. Cox, P. Cairns, S. Dhoparee, A. Epps, and T. Tijis. Measuring and defining the experience of immersion in games. *International Journal of Human-Computer Studies*, 66:641–661, 2008.

[6] J. Lauriere. A language and a program for stating and solving combinatorial problems. *Artificial Intelligence*, 10:29–127, 1978.

[7] R. L. Mandryk and K. M. Inkpen. Physiological indicators for the evaluation of co-located collaborative play. *Proceedings of the ACM Conference on Computer Supported Cooperative Work*, pages 102–111, 2004.

[8] J.-C. Régin. Generalized arc consistency for global cardinality constraint. In *AAAI/IAAI, Vol. 1*, pages 209–215, 1996.

[9] I. Shlyakhter. Generating effective symmetry-breaking predicates for search problems. *Discrete Appl. Math.*, 155(12):1539–1548, 2007.

[10] H. Simonis. Sudoku as a constraint problem. *Proceedings of the Modelling Workshop at CP*, 2005.

[11] H. Simonis. Kakuro as a constraint problem. *Proceedings of the Modref Workshop at CP*, 2008.

[12] H. Simonis. Shikaku as a constraint problem. *CSCLP 2009*, 2009.

[13] B. M. Smith, K. E. Petrie, and I. P. Gent. Models and symmetry breaking for peaceable armies of queens. *Proceedings of CPAIOR*, Springer LNCS 3011:271–286, 2004.