Solving SameGame and its Chessboard Variant

Frank W. Takes Walter A. Kosters

Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands

Abstract

We introduce a new solving method for SameGame puzzles based on Monte-Carlo simulation, that improves the previous best known method. We also cover the chessboard variant, a subclass of the SameGame puzzle for which we can exactly define which puzzles are solvable and which ones are not. Next to this we present a quantitative approach to classify the difficulty of a given puzzle, as well as an indication of whether or not a given puzzle is solvable.

1 Introduction

This paper is about the puzzle (or game) *SameGame*, also known as Clickomania or Jawbreaker. For computer scientists in the field of Artificial Intelligence [7] and Combinatorial Game Theory [3], games that have associated NP-complete decision problems [6], like SameGame, remain an interesting field of study. The search space is way too large and too dynamic for traditional search methods, making the search for an optimal solution an extremely hard task. Compared to games such as Tetris and even Sokoban, SameGame has not been studied much, and there are still many open questions. We will attempt to shed some light on this subject, answer several questions, and present some interesting facts about SameGame. We also give an outline of possible search methods that could be applied, as well as a new algorithm for solving SameGame puzzles that improves the algorithm presented in [8].

The paper is organized as follows. In Section 2 we explain SameGame, and describe some properties. We pay special attention to "chessboard" like variants in Section 3. In Section 4 we describe our algorithm, and we mention results in Section 5. Section 6 concludes.

2 SameGame

SameGame is a single player game (or puzzle) that has seen a big increase in popularity over the past few years. It has actually been around since 1985, when it was invented in Japan by Kuniaki Moribe. It was first released in the western world under the name Chain Shot, but has been redistributed (often with slightly different rules) under, amongst others, the names Clickomania, HMaki, Samegame, Jawbreaker, Bubblets, and Bubble Breaker. We will refer to it as *SameGame*.

2.1 Rules

The game is played on a two-dimensional *board* (or grid) with N *tiles* (or fields, squares, stones or blocks). The tiles of a puzzle each have one out of C different *colors*. Figure 1 shows two puzzles with $N = 225$ (the board is of size 15×15) and $C = 5$ (red, green, blue, purple and yellow). If we talk about a certain tile on some board with height h and width w, we denote its position by (i, j), with $1 \leq i \leq h$ and $1 \leq j \leq w$. To clarify, tile $(1, 1)$ is situated in the top left corner. Initially, the board is usually filled with randomly colored tiles. Two tiles are directly *adjacent* if they are horizontal or vertical direct "neighbors". A tile (i, j) not on the edge of the board has neighbors $(i - 1, j)$, $(i + 1, j)$, $(i, j - 1)$ and $(i, j + 1)$, while a tile on the edge has only three of these neighbors, and a corner tile only two. A *group* is defined as a set of *two or more* tiles of the same color, where it is possible to go from any of its tiles to any other by repeatedly visiting adjacent tiles; or briefly: a group is connected. A tile at the board that does not belong to any group is called

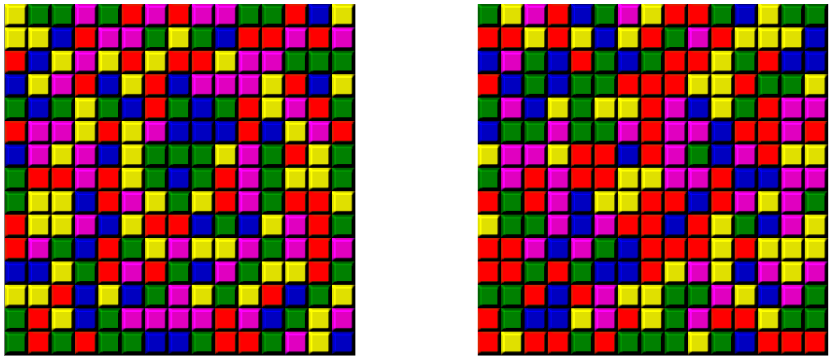

Figure 1: Two puzzles, no. 1 (left) and no. 9 (right) from the benchmark set at [5]. Solution for puzzle 9: 12,6 - 5,11 - 12,5 - 14,6 - 12,11 - 8,12 - 10,2 - 6,1 - 12,14 - 8,1 - 6,12 - 9,14 - 9,6* - 7,4 - 5,13 - 7,3 - 11,7 - 13,14* - 14,2 - 14,3 - 14,14 - 12,5 - 10,11 - 9,12 - 8,3 - 15,7* - 13,13 - 15,8* - 6,8 - 15,10 - 14,13 - 13,10 - 12,2* - 12,9* - 11,1 - 13,4 - 15,1** - 15,3 - 15,1 - 15,7 - 15,5 - 15,2* - 13,1 - 14,2 - 15,2 - 14,1 - 15,1 (the bottom leftmost tile of a group is chosen, a * denotes a big group, ** denotes the biggest group)

a *singleton*. A *move* consists of deleting a group, after which several things can happen, depending on the type of puzzle:

- *Vertical gravity and column shifting*: tiles that are above the deleted tiles fall down. As soon as an entire column is empty, the columns to the right of the empty column(s) shift to the left.
- *Vertical and horizontal gravity*: tiles that are above the deleted tiles fall down. After that, tiles that are to the right of empty positions caused by the deletion or the previous falling down of tiles are shifted to the left (they are drawn towards the left side of the board).

When several groups have been removed, and either one of the two options above has done its work, one or more empty columns can arise on the rightmost side. Now two things can happen:

- The empty columns *remain empty*.
- The rightmost column is repeatedly filled with a fresh column of *new random tiles*, that then again behaves in either of the two ways described above.

Clearly, the first version of the last two above allows us to do a full-knowledge deterministic examination, while the second version can theoretically go on infinitely.

The game can have several *goals*. One obvious goal could be to empty the board, leaving no tiles. We call a puzzle *solvable* if the board can be emptied by doing a series of moves. Another goal is to get a score as high as possible. In most scoring schemes the score depends on the size of the removed groups, where it usually grows quadratically with the size of the removed group. Often bonus points are also awarded for emptying the board, and points are subtracted for leaving tiles on the board. Bubble Breaker uses $n(n-1)$ as reward, and other versions use $(n-2)^2$, where n is for the number of blocks in a group that is removed. The functions have the common property that removing one large group of size r always gives a higher score than removing two groups of p and q blocks, where $p+q=r$ and $1 \leq p, q \leq r-1$.

The game *ends* when the board is empty or no more groups can be removed. If new random columns are added, the game ends when the entire board and especially the entire bottom row is filled with singletons.

From now on, we assume that we talk about the version of SameGame where there is vertical gravity and column shifting, and no random tiles are added once a column has been emptied. So we are now talking about a fully deterministic finite problem, with full information. We use the $(n-2)^2$ scoring function.

2.2 Complexity

In [1] it is proven that the problem of determining whether or not an initially filled board can be emptied is NP-complete. Obviously, finding the path with the maximum score is just as hard, as remarked by [8]. A problem in SameGame is the large search space and a high and varying branching factor: the amount of groups in a certain state. The puzzles in Figure 1 start with a branching factor of around 40. An average solution takes about 45 moves, so one can easily infer that the number of possible states is immensely large.

Another problem is the complexity of the changes that can occur to the board. Removing a group in one state can cause the total set of groups to drastically change. Groups can "suddenly" appear, merge, split or even disappear completely. This makes it hard to predict how good a certain move is, as doing a move can change the entire layout of the board, perhaps making any scoring at all impossible, or present a move with an extremely high score in the next state.

2.3 Solvability

It is interesting to see how many puzzles actually *are* solvable. For smaller board sizes and a small amount of colors we can calculate this value by simply generating all possible puzzles, and determine brute-force (for details on solving methods, see Section 4) if they are solvable or not. With two colors ($C = 2$), up to $N = 25$ (boards of size around 5×5), this is still doable with the brute-force method. For bigger board sizes, we can no longer exactly determine this value. However, by sampling P random puzzles, we can still get a good estimate of the solvability. Table 1 shows an overview of how many puzzles are actually solvable for the two-color version of SameGame. Values in *italics* are sampled puzzles with $P = 100{,}000$ samples. Obviously, the conclusion we can draw is that the bigger the board, the bigger the chance the puzzle is actually solvable. This appears to apply for $C = 3$ as well, see [10]. Also, for puzzles of size 15×15, with $C = 5$, there appears to be a quite big chance that the puzzle is actually solvable — see Section 5.

C=2	*W*=1	*W*=2	*W*=3	*W*=4	*W*=5	*W*=6	*W*=7	*W*=8	*W*=9	*W*=10
H=1	0.0%	50.0%	25.0%	37.5%	37.5%	40.5%	45.3%	49.5%	54.4%	59.0%
H=2	50.0%	37.5%	59.3%	66.3%	71.5%	76.6%	81.9%	86.4%	89.3%	91.9%
H=3	25.0%	59.3%	77.3%	81.6%	85.4%	89.0%	91.8%	*94.0%*	*95.4%*	*96.7%*
H=4	37.5%	68.7%	82.1%	86.8%	90.6%	*93.3%*	*94.8%*	*97.3%*	*97.5%*	*98.4%*
H=5	37.5%	76.6%	88.0%	92.1%	*94.8%*	*96.7%*	*97.6%*	*98.4%*	*98.9%*	*99.3%*
H=6	40.5%	82.4%	92.5%	*95.3%*	*97.2%*	*98.3%*	*98.9%*	*99.1%*	*99.6%*	*99.7%*
H=7	45.3%	88.0%	*95.5%*	*97.5%*	*98.5%*	*99.2%*	*99.5%*	*99.7%*	*99.8%*	*99.9%*
H=8	49.5%	92.0%	*97.0%*	*98.6%*	*99.2%*	*99.6%*	*99.8%*	*99.9%*	*99.9%*	*99.9%*
H=9	54.4%	94.5%	*98.3%*	*99.3%*	*99.6%*	*99.8%*	*99.9%*	*99.9%*	*99.9%*	*99.9%*
H=10	59.0%	96.5%	*99.0%*	*99.5%*	*99.8%*	*99.9%*	*99.9%*	*99.9%*	*99.9%*	*99.9%*

Table 1: Percentage of puzzles of size $H \times W$ with two colors that is solvable.

2.4 Difficulty

It is hard to grasp the difficulty of a SameGame puzzle. Figure 1 shows two puzzles of which the left puzzle has an equally distributed amount of each color, where the right one has red dominating, and already some bigger groups formed. Therefore, the right puzzle seems easier. We could get a more realistic estimate of the difficulty by generating $P = 1{,}000{,}000$ random samples, and determining how the scores are distributed. We tested two types of solution methods, random and random with a bias towards forming a large group of one color. The latter means that during the solution, the groups of the color of which there are the most tiles left, are not touched until not possible otherwise. The results are shown in Figure 2. Regardless of the method used, each puzzle shows a similar graph. The peak indicates the average score one would obtain when playing the puzzle (either random, or with a bias towards one color). For puzzle 9, for both methods, the peak clearly lies at a higher score than for puzzle 1; we can conclude that puzzle 9 is easier.

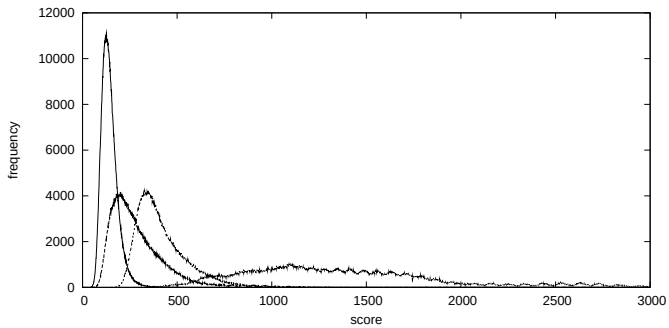

Figure 2: The frequency of scores obtained by doing $P = 1{,}000{,}000$ random simulations. First and second peak: puzzle 1, random, resp. random with bias simulation; third and fourth peak: puzzle 9, idem.

3 Chessboard Variant

In this section we will discuss a subset of SameGame puzzles, the chessboard and its variations:

Definition 1 *A **chessboard variant** of a SameGame puzzle, or briefly a **chessboard**, has two colors ($C = 2$; black and white) and each tile has a color different from that of its neighbor, meaning that both horizontally and vertically, colors interchange.*

The chessboard itself is clearly not solvable, as we cannot perform any moves because there are no groups to be removed. However, if we invert the bottom rightmost tile of the chessboard, the puzzle suddenly becomes solvable, as shown in Figure 3.

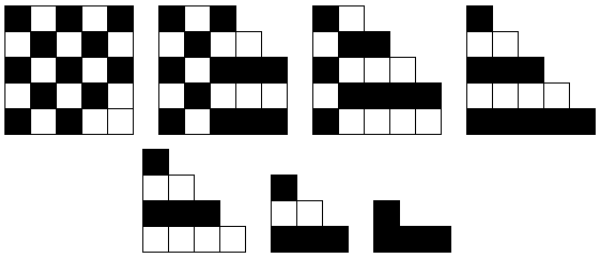

Figure 3: A puzzle of size 5×5 with $(5, 5)$ inverted is solvable.

Theorem 1 *Any chessboard variant of a SameGame puzzle of size $h \times w$ with $h, w \geq 4$, and position (h, w) inverted, is solvable.*

Proof. We start by removing the only group that *can* be removed, at position (h, w). We then keep removing the group at the bottom rightmost position, which will first repeatedly be (h, w), after which it will be $(h, w - 1)$, $(h, w - 2)$, etc., until we reach the point where we want to remove $(h, 3)$. Now we first remove $(h - 1, 1)$, after which we remove $(h, 3)$ and we are done, cf. Figure 3. □

Even more interesting is the fact that *any* chessboard of 5×5 or bigger (in both dimensions) with exactly one mistake in the *lower half* of the board (in case of an odd board height, the part of the board below and including the middle row), is solvable, see [10]. We will concentrate on the complementary situation, which happens to be a consequence of a more general theorem presented below:

Theorem 2 *Any chessboard variant of a SameGame puzzle of size $h \times w$ with $h, w \geq 5$, and exactly one position (i, j) with $1 \leq i \leq \lfloor h/2 \rfloor$ and $1 \leq j \leq w$ inverted, is not solvable.*

For one-column SameGame, in [1] it is already proven that if a puzzle has a checkerboard (in this case identical to a chessboard) longer than half the total number of groups, then the puzzle is unsolvable. We prove a similar property for two-dimensional chessboards with respect to the board size. We therefore generalize our previous theorem and formulate the main result of this section, along with a crucial lemma:

Theorem 3 *A two-color SameGame puzzle of size $h \times w$ with $h, w \geq 5$ where the strict lower half (every position (i, j) with $i \geq \lceil h/2 \rceil$) consists of a chessboard, and the upper half of arbitrary tiles (perhaps even with tiles omitted on top of some columns), is not solvable.*

Lemma 1 *If in two-color SameGame we remove a group which includes position (i, j), but which does not include $(i + 1, j)$, we know that position (i, j) will either remain empty, or will be filled with a tile of the same color as $(i + 1, j)$.* □

Proof of Theorem 3. We will give a sketch of the proof. We will show that it always holds that in every column, there are strictly more tiles that were part of the chessboard than tiles that were part of the "upper half". This property is referred to as the *size property*. It certainly holds in the beginning, see Figure 4(a). After a while, the border looks like that from Figure 4(b). Here a '-' denotes absence of a tile (also called "empty", the whole column above this tile also being empty), a 'W' stands for empty or a white tile, and a 'B' denotes empty or a black tile. A '?' will be used when we cannot infer anything yet. In the sequel we will define three distinguished possible constituents of the border, already visible in Figure 4: the *abyss*, the *stairway* and the *plateau*.

We now first note that the removal of a group entirely contained in the "upper half" of course does not violate the size property. Next we examine removal of tiles from the chessboard part. We distinguish

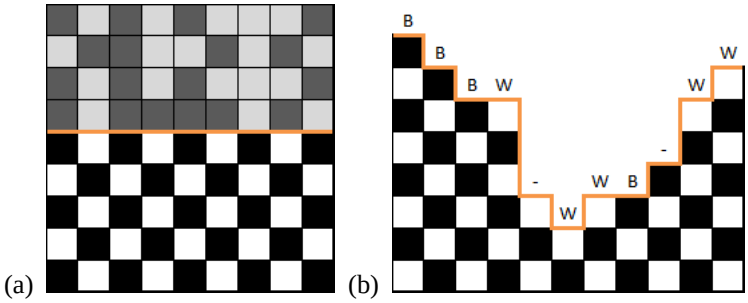

Figure 4: (a) The original border between the black/white chessboard and the greyscale "arbitrary half" for a 9×9 puzzle. (b) The chessboard after performing several moves (for a somewhat larger board).

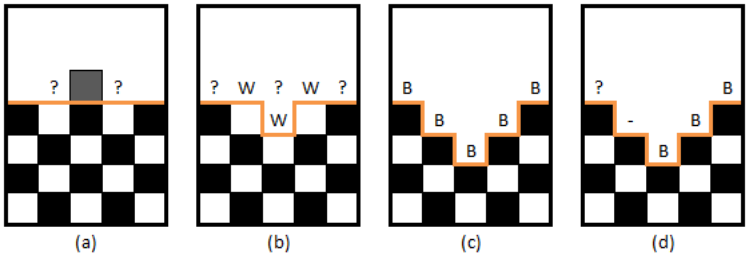

Figure 5: Removing tiles from the top row of the chessboard.

between tiles from its top row, and others. Indeed, thanks to Lemma 1, we know that for the latter ones there is a tile of the same color (or no tile at all) above; for tiles from the top row this does not necessarily hold.

The first tile that is removed from the chessboard is always removed from its top row, as displayed in Figure 5(a). This does not hurt the size property. After this move we know that the tile in the newly created "hole" is filled with a white tile or remains empty because of Lemma 1. We know the same applies for the two neighbors of the removed group. In Figure 5(b) they are indicated by a 'W': they were either white or empty in the first place, or they were black and removed in the previous step. In that latter case, again because of Lemma 1, they are now white or empty.

Now for a next possible step, if the two neighbors were both actually white and removed, the situation is now that of Figure 5(c). It could also have happened that one or both of the neighbors were empty. The case where the left neighbor was empty is displayed in Figure 5(d). In every column of the part of the chessboard that we removed, the size property is maintained. The '?' at the top left of Figure 5(d) is not a problem. If we remove it from the top, it does not hurt our property. We will never remove it from the right side because there is an abyss (see below) there, and if we remove it from the left side it will perform analogously to the situation in Figure 5(a) and cause no problems either.

After some time the border of the chessboard looks like the one in Figure 4(b). Notice that because of Lemma 1, we always exactly know what is on top of a tile at the border — except perhaps for some tiles from the top row of the chessboard in a plateau. The border consists of three types of situations:

1. A horizontal *plateau*. We know how this behaves, as we know exactly what is on top of it, again with the "harmless" special situation in the remainder of the top row of the chessboard.

2. A *stairway* (stair-depth of at most 1). A stairway is either bounded by a plateau (top left of Figure 6(a), or by an abyss. Observe that the stairway could have an arbitrary length between A and D, and that it could also consist of just tile A with an abyss or horizontal line of the chessboard at its end.
   Figure 6 shows that after removing the part of the stairway consisting of tiles A, B, C and D, the positions A, B, C and D are always filled with either a white tile or nothing because of Lemma 1. So we know that if we remove a stairway, either a stairway remains, or an abyss or plateau appears.

3. An *abyss* (stair-depth of 2 or more). This type of situation has a '-' in Figure 6(c): if this occurs, then the bottom of the abyss is always empty. To make the abyss deeper, only the tile below the bottom of the abyss can be removed, which will always have to happen from the side.

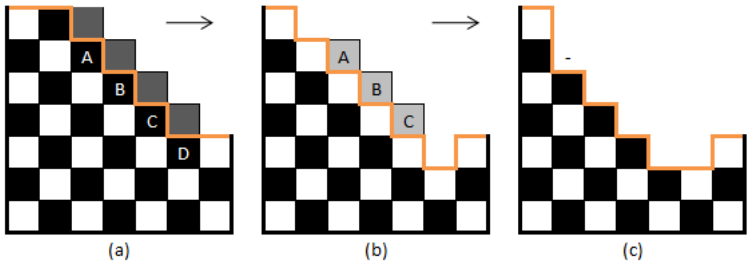

Figure 6: Characteristic positions of the chessboard: “plateau”, “stairway” and “abyss” (’-’ at the bottom).

Notice that at all times the boundary of the chessboard can easily be detected. We have shown that the size property is always maintained, hence tiles from the bottom row can never be removed. This therefore proves Theorem 3, and also Theorem 2. □

We note that we can always only remove the topmost tile of a column of the chessboard part of the puzzle. Furthermore, we can never create an abyss of depth ≥ 2 and width 1 in the chessboard: an abyss of depth k has at the bottom also width k.

4 Solving Methods

A *Brute Force* approach is not an option for SameGame, concerning the huge search space size. *Best-first search methods* such as *greedy search* will do no good in SameGame either. Methods like *Branch and Bound* and *(ID)A** as described in [7] require an admissible evaluation function. In order for the search to be complete, they need a good over-estimator (upper bound) for the quality of a certain state.

A possible upper bound for the quality of a SameGame position could be the current score plus the score gained pretending that all the remaining colors are connected. However, using this bound is not as efficient as one may think. In a child node of some current node, the upper bound will be much lower than that of all the nodes at the level of its parent, unless we previously removed a really big group, resulting in something which will often closely resemble the undesirable Breadth First Search (BFS). The upper bound can however still be used to prune our search tree. Other than that we can include the $1,000$ bonus points in our evaluation function, and omit them if there is a color of which there is only one tile left, because these situations will, similar to the deadlocks in Sokoban from [9], never allow us to empty the board. Deriving a better strict upper bound seems hard.

Beam search keeps a “beam” of candidates, all at the same level of the search tree. In each iteration the best B (where B is the “beam size”) children are kept and form the new beam. Any time we reach a leaf, we check if it is better than the current best solution. The advantage of this method is that the complexity is at most $B * d$, where d is the maximum depth reached. Beam Search does not perform a complete search, so we could also settle for a better estimator than an over-estimator. Though not perfect, we could for example take our current score so far, perhaps incremented by the score obtained when removing all current groups.

The *Banker’s Algorithm* assigns a certain budget to each child of the root node with which that node is “allowed” to explore the part of the tree below it. When the search reaches a depth where its budget has been spent, a greedy simulation can be performed. A variant of the Banker’s algorithm is used by [2] in his SameGame algorithm called Depth-Budgeted Search (DBS).

Monte Carlo (MC) tree search algorithms are a form of best-first search algorithms that do not evaluate the quality of a node based on some evaluation function. Instead, Monte Carlo bases the quality of a node on a number of random games, say R, that are played for each of the nodes in the list of candidates. The “best” node is then chosen as successor. The evaluation of which node is the best, depends on the type of problem and the behavior of the search space. In [8] a Monte Carlo algorithm for SameGame called SP-MCTS is implemented, which is based on Upper bound Confidence Trees (UCT) [4].

In SameGame we could maximize groups by not touching a certain color, e.g., the dominant one, hoping to form a big group of the color that we are not touching. This approach has several disadvantages, such as the fact that a lot of useless moves are performed before the one big group is removed, and that this method will not consider forming several other larger groups.

We propose using *Monte Carlo with Roulette-Wheel Selection (MC-RWS)*, a method that not only tries to maximize one group of a certain color, but also tries to create bigger groups of another color. The Monte

Carlo part of our algorithm is as follows: to evaluate which child we choose as successor of the current node, we pick the node with the highest average of R (e.g., $R = 1,000$) simulations. The strength lies in the way in which we perform these random simulations.

Let us assume we have a set of colors $\{c_1, c_2, \ldots, c_C\}$ and a function $f(c)$ that returns the amount of tiles that are left of color c. Let α define how big our focus on larger groups is; large α values favor removal of groups of non dominating colors. During a simulation in MC-RWS, the chance $P(c_i)$ of selecting a group of color c_i ($1 \leq i \leq C$) to be removed next is equal to:

$$P(c_i) = \frac{1}{C-1}\left(1 - \frac{(f(c_i) - \theta)^\alpha}{\sum_{k=1}^{C}(f(c_k) - \theta)^\alpha}\right)$$

Inspired by the cooling schemes used in Simulated Annealing [7], the results turned out best when using a linear scheme, where α depends on the amount of tiles left: $\alpha = 1 + (\beta/N) * \sum_{k=1}^{C} c_k$ (N is the total amount of initial tiles). Here β defines how big our focus on bigger groups actually is. A value of 4 for β turned out to work fine for puzzles of size $N = 15 \times 15 = 225$. Threshold θ is used to prevent problems with large values, and could for example be set to the size of the smallest group divided by 2.

Because forming the largest group is still very important, at first the most prominent color will barely be played. As the puzzle evolves, the focus on forming groups of other colors becomes larger as well, resulting in multiple bigger groups, meanwhile still building one large group. An example of a solution with multiple big(ger) groups is the one of puzzle 9 in Figure 1.

5 Results

We have tested the different methods on the standardized test set available at [5]. The set consists of 20 15×15 puzzles with 5 colors. Games 1–10 are randomly distributed, 11–15 have 45 fields per color and games 16–20 have one dominating color. The maximum computation time used for the algorithms is 2 hours on a 3,2 GHz Quad-Core machine with 6 GB memory (though memory usage is extremely limited).

Puzzle	Random	MC-Biggest	MC-Avg	MC-Avg-Bias	DBS	SP-MCTS	MC-RWS
1	443	723	1,771	2,145	2,061	2,557	**2,633**
2	831	1,369	2,585	3,545	3,513	3,749	**3,755**
3	641	1,497	2,481	2,681	3,151	3,085	**3,167**
4	709	1,521	3,247	3,749	3,653	3,641	**3,795**
5	533	1,601	2,641	3,687	3,093	3,653	**3,943**
6	903	2,393	3,013	3,917	4,101	3,971	**4,179**
7	487	947	2,271	2,731	2,507	2,797	**2,971**
8	1,347	2,291	3,433	3,847	3,819	3,715	**3,935**
9	1,385	2,731	4,189	4,421	4,649	4,603	**4,707**
10	459	1,335	2,585	3,097	3,199	3,213	**3,239**
11	491	1,219	2,167	2,667	2,911	3,047	**3,327**
12	689	1,607	2,535	2,977	2,979	3,131	**3,281**
13	613	1,165	2,283	2,917	3,209	3,097	**3,379**
14	575	1,209	2,401	2,597	2,685	2,859	**2,697**
15	839	1,567	2,831	3,199	3,259	3,183	**3,399**
16	2,267	3,509	4,533	4,613	4,765	4,879	**4,935**
17	817	2,837	3,355	4,643	4,447	4,609	**4,737**
18	1,167	3,239	4,497	4,855	5,099	4,853	**5,133**
19	1,471	2,041	3,323	4,565	4,865	4,503	**4,903**
20	943	2,855	2,811	4,469	4,851	4,853	**4,649**
Total	17,610	37,656	58,952	71,322	72,816	73,998	**76,764**

Table 2: Comparing scores on the benchmark set at [5].

The table above shows how the different search methods score on the test set, where the first column corresponds to the number of the puzzle. The second column is the best score acquired by playing one million random games. The total score acquired is only $17,610$ in total. The third column, MC-Biggest, is a Monte Carlo method that evaluates the children of the current node in the search tree by sending $R = 1{,}000$ random probes down to the leaf nodes, and then picks the child with the highest score to again perform the same process, until a leaf node is reached. Instead of taking the child with the best random probe as a successor, we can also choose to take the node of which the $R = 1{,}000$ random probes produce the highest *average* score, displayed in column MC-Avg. This improvement gets us roughly another $21,000$ points, partly thanks to the $1,000$ bonus points awarded for emptying the board on all 20 puzzles.

The methods discussed so far use no domain-knowledge whatsoever. We know that in SameGame, larger groups give considerably higher scores. If we incorporate this bias into our MC-Avg algorithm, the total score again increases by about $12,000$ points (column MC-Avg-Bias) to a score comparable to SP-MCTS, the algorithm described in [8]. The limitation of the latter two techniques is that the focus is on one group, and that color is not played unless absolutely impossible otherwise. This does not directly maximize the score of the groups of the four other colors that are removed along the way. To more or less enforce this, we have presented MC-RWS in Section 4. This method outperforms SP-MCTS on 18 out of 20 puzzles by a total of $2,766$ points, obtaining a total score of $76,764$. The actual solutions can be found in [10].

The column titled "DBS" presents the score of the variation of the Banker's Algorithm implemented by [2] which was the leading algorithm before SP-MCTS. MC-RWS has outperformed this algorithm on 18 out of 20 puzzles. An anonymous competitor known as "spurious_ai" has achieved a total score of $82,604$ on the test set by means of an algorithm; however, not much is known about this algorithm other than that it uses some kind of Beam Search, employs many GBs of memory, and exploits a multi-processor architecture.

6 Conclusions and Future Work

We have outlined some quantitative and qualitative properties of SameGame puzzles. We presented a quantitative approach to classifying the solvability and difficulty of a SameGame puzzle and examined the chessboard and its variants, for which we can exactly determine whether or not they are solvable. We also developed a promising method called Monte Carlo with Roulette Wheel Selection (MC-RWS) that not only tries to maximize one group of a certain dominant color, but also tries to form large groups of the other colors.

Future work could include attempting to apply MC-RWS to other similar games, as well as finding additional features characterizing always (un)solvable types of puzzles such as the chessboard. Another interesting question could be to approximate the amount of solutions that a certain puzzle has. A further examination of the initial moves might also be of interest. Finally, the question of whether or not for example two-color SameGame is NP-complete still remains an open question.

References

[1] T. Biedl, E. Demaine, M. Demaine, R. Fleischer, L. Jacobsen and J. Ian Munro. The complexity of Clickomania. In *More Games of No Chance*, pages 389–404, Cambridge University Press, 2002.

[2] D. Billings. GAMES group, University of Alberta. E-mail conversation, October 2008.

[3] E.D. Demaine and R.A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In *Games of No Chance 3*, pages 3–56, Mathematical Sciences Research Institute Publications, volume 56, Cambridge University Press, 2009.

[4] L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *Proceedings of the 17th European Conference on Machine Learning*, pages 282-293, LNCS/LNAI 4212, Springer, 2006.

[5] S. Misch and A. Schulze. Website js-games.de [accessed June 11, 2009], `http://www.js-games.de/eng/games/samegame/lx/play`.

[6] G. Kendall, A. Parkes and K. Spoerer. A survey of NP-complete puzzles. *International Computer Games Association Journal* 31(1): 13–34, 2008.

[7] S.J. Russell and P. Norvig. *Artificial Intelligence, A Modern Approach*. Second edition, Pearson Education, 2003.

[8] M.P.D. Schadd, M.H.M. Winands, H.J. van den Herik and H. Aldewereld. Addressing NP-complete puzzles with Monte-Carlo methods. In *Proceedings of the AISB 2008 Symposium on Logic and the Simulation of Interaction and Reasoning, Volume 9*, pages 55–61, The Society for the Study of Artificial Intelligence and Simulation of Behaviour, 2008.

[9] F.W. Takes. Sokoban: Reversed solving. In *Proceedings of the 2nd NSVKI Student Conference*, pages 31–36, Utrecht, The Netherlands, 2008.

[10] F.W. Takes. *Solving SameGame and its Chessboard Variant*. Research thesis, Leiden University, The Netherlands, 2009, `http://www.liacs.nl/~ftakes/samegame/`.