Breaking the Code

brENIAC – an Artificial Mastermind Player

Gil Dekel, Sarah Mathew, Ahmed Shah, and Jaspal Singh

Abstract

Mastermind is a two-player code breaking game. The players decide in advance how many games they will play. One player chooses a pattern of colors and the other player must guess it. The code-breaker player has a limited number of guesses and a time limit to break a code. The game becomes substantially more complex as we increase the number of possible colors and the length of the code. Our goal is to develop an artificial Mastermind code-breaker that can play a game of any size in a reasonable manner, which includes good decision-making and performance.

I. Introduction and Background

Mastermind is a two-player code breaking game in which one player comes up with a secret pattern of colors of a certain length and the other player must guess it. A guess can break the code, lose the game (if the guess limit is met) or, as in most cases, get a response. Each response states how many of the guessed colors are in the correct position and how many correct colors are in the wrong position. This information is returned as a pair of numbers which are called 'Bulls' and 'Cows' (*B,C*) respectively. The game's complexity increases as the length of the patter and the number of colors increases. More specifically, the number of possible guesses (codes) in any game configuration (*P,C*) is C^P, where *C* is the number of colors and *P* is the number of pegs (i.e. the length of the code). In order to construct an artificial player that can play games of any size in a reasonable manner, we first reviewed previous efforts in solving Mastermind.

Donald Knuth determined a Five-guess algorithm that would solve a four-peg six-color game (4,6) in five guesses or fewer. Knuth's algorithm utilizes set theory and is based on constructing the set of all the possible codes in a given game configuration. For a (4,6) problem size, this would result in a set of 6^4 = 1296 possible solutions. Knuth's method uses a greedy strategy that aims to minimize the number of possible solutions at each step (1976) and his algorithm produces an efficient solution for a (4,6) problem. However, games of size (26,26) produce a colossal solution space (26^{26}) that, in most cases, cannot be constructed by and contained within common computers. Therefore Knuth's algorithm does not scale well with the problem size. Another approach is generic algorithms. This family of algorithms requires a strong evaluation function and the ability to "evolve" over time. With each iteration the algorithm chooses the best performing guesses and through recombination and mutation, produces the next generation of guesses. This process repeats until the code is broken (Merelo, Mora, Cotta, & Fernández-Leiva 2013). However, similar to normal evolution, these algorithms require many iterations (or guesses in this case) to succeed. Given that, in our case, we are limited to 100 guesses per code, we decided against genetics algorithms.

Our initial approach was to implement a player that guesses monochromatically (i.e. all As, then all Bs, etc.) and then exhaustively enumerates through the remaining solution space. Monochromatic elimination helps to reduce the solution space by determining the colors in *C-1* guesses. However, for a game with 26 pegs and 26 colors, exhaustively enumerating the remaining solution space once monochromatic elimination is complete would not break the code in under 75 guesses (worst case 26 unique colors ordered in reverse Z-A; 25 monochromatic guesses required).

II. Method

1. Guess Construction and Internal State Representation

A better monochromatic elimination approach is described by Rao, in which each monochromatic guess also includes a target color. In addition, Rao's player maintains an inner representation of its environment and updates it according to each guess' response (1982). Our player is based on Rao's approach. It constructs its guesses using a similar color ordering strategy and updates an inner state.

Table 1. A response to a guess is processed according to six rules. '#' stands for 'any number'. Once all targets are identified, the player ceases to execute the parts in which new targets are pushed to *T* and the different TPLs.

	Response	Analysis of Guess
		T = Ø
1	**(0, 0)**	Current filler is not in the code • Remove from *F*
2	**(#, 0)**	Identified # targets with color = filler • Push the filler to *T* # times • Enter the new target # times to the TPL • Remove from *F*
		T ≠ Ø
3	**(0, 1)**	Filler does not exist • Remove from *F* • Pop the target's TPL (code does not have current target in this peg)
4	**(#, 0)**	Solved target • Target is locked with location • Remove the locked location from all other targets in the TPL • Add (#-1) fillers as targets to *T* and TPL
5	**(#, 1)**	Identified # targets with color = filler • Push the filler to *T* # times • Enter the new target # times to the TPL • Remove from *F* • Pop target's TPL and remove that location from fillers' TPLs
6	**(#, 2)**	Target is masking the filler • Filler is locked with location • Remove the locked location from all other targets in the TPL • Add # fillers as targets to *T* and TPL

The inner state of the player is constructed of a 'filler' queue *F* that is initialized with all *C* colors and an initially empty 'target' queue *T*. A target is any color that is known to be in the code but its location is undetermined. A filler is any color from the color list that its contribution to the solution is as-yet-unknown (may or may not be in the code). Every time a new target is discovered, it is added to the *Targets' Possible Locations* (TPL) list. The TPL is a list of lists. In each list, the first element is the target's color and the second element is another list of its possible locations. For example, in a game size of (4,6) the TPL might hold

(((A (1 2 3 4)) (B (1 2 3 4)) (C (1 2 3 4)))

where A, B, and C are known targets, *T* = (A B C), and *F* = (D E F). This means the player did not lock-in any targets yet and it is still on the lookout for the fourth (and last) target.

Our player oscillates between two main general states (not to confuse with inner state). It either knows of at least one target (**T≠ Ø**), or it doesn't (**T= Ø**). When T is empty, guesses are produced in a monochromatic manner and the order in which colors are tested is determined by the *F* queue. As soon as the first target is found, the player switches to a different strategy in which it pops a target from *T* and a filler from *F*, inserts the target at its next possible position from its TPL, and fills the remaining pegs with the filler color. In addition, whether or not a filler was added as a target to *T*, it is removed from *F*. When a target is chosen, it is not replaced until its position is determined. On the other hand, fillers are replaced with every guess until the filler list is empty or the number of known targets is equal to the number of pegs. Once all the targets are identified, the player keeps focusing on one target at a time, however, since the filler list is now empty, it chooses a substitute filler from the target list and sticks with it until either target or filler are locked in.

Before the player processes a response, the number of *solved* pegs is subtracted from the number of bulls returned. Because of the unique order in which guesses are made, negative results are not a concern. Our player is processing responses according to the six rules in Table 1. Each response results in a notable reduction of the solution space by either reducing the branching factor (when a color is eliminated), reducing the power's magnitude (when a peg is solved), identifying new targets, or removing locations from target's TPLs. Another important feature of our player is the ability to detect when a target's TPL set is reduced to 1 indirectly. This happens eve-

Table 2 – Tracing through an example of size (8,10) using the guess response table

	H	C	H	G	F	C	A	G	Response – (# locked, 0)		Rule Applied
1	A	A	A	A	A	A	A	A	(1, 0) – (0, 0)	(1, 0)	2
2	A	B	B	B	B	B	B	B	(0, 1) – (0, 0)	(0, 1)	3
3	C	A	C	C	C	C	C	C	(1, 2) – (0, 0)	(1, 2)	6
4	D	**C**	A	D	D	D	D	D	(1, 1) – (1, 0)	(0, 1)	3
5	E	**C**	E	A	E	E	E	E	(1, 1) – (1, 0)	(0, 1)	3
6	F	**C**	F	F	A	F	F	F	(1, 2) – (1, 0)	(0, 2)	6
7	G	**C**	G	G	**F**	A	G	G	(4, 1) – (2, 0)	(2, 1)	5
8	H	**C**	H	H	**F**	H	A	H	(5, 0) – (2, 0)	➔ (3, 0)	4
9	C	**C**	G	G	**F**	G	**A**	G	(5, 1) – (3, 0)	(2, 1)	5
10	G	**C**	C	H	**F**	H	**A**	H	(5, 1) – (3, 0)	(2, 1)	5
11	G	**C**	G	C	**F**	G	**A**	G	(4, 2) – (3, 0)	(1, 2)	6
			← Arc Consistency →								Auto
12	**H**	**C**	**H**	**G**	**F**	**C**	**A**	**H**	**W**	-	-

ry time the position of a locked is removed from all other TPLs, and those with two remaining possible locations are reduced to one. The affected targets are then added to a 'lock-in' queue and the process repeats until the queue is empty. This is, in fact, applying arc-consistency over the TPL list until all targets with |TLP|=1 are locked-in.

Using our approach, Table 2 tracks a 12-guess solution to a problem of size 8-peg 10-color (i.e. 10^8 = *100 million* legal guesses). By the 4th guess, our player identifies that the code consists of one A, no Bs, two Cs (one of which it immediately locked in at peg 2), and that there are no Ds. By the 8th guess, the player knows that there are no Es, one F at peg 5, two Gs, and about to discover A's position. At the 11th guess, we learn that the current target C is masking one of the two Gs, so the player locks in G and triggers a terminal sequence of arc-consistency. The next and final guess breaks the code.

2. Theoretical Analysis

For ease of presentation and without loss of generality, consider a 4-peg 2-color example. The initial search space $|S| = C^P$ ($2^4 = 16$) is fully enumerated in Table 3. As stated earlier, our player reduces the search space with every response. The three following scenarios describe the reduction.

Peg Lock-ins: When the player solves a peg, the number of pegs to solve is reduced by one, thus reducing the magnitude of the power P by 1. The search space is then reduced by:

$$|S| - C^{P-1} = 16 - 2^{4-1} = 16 - 8 = \mathbf{8}$$

possible solutions. If the player locks-in the first A in the sequence, codes 9 to 16 in the space described in Table 3 are removed and the player will never attempt those as a guess.

Tables 3. Search space for (4,2)

1	A	A	A	A
2	A	A	A	B
3	A	A	B	A
4	A	A	B	B
5	A	B	A	A
6	A	B	A	B
7	A	B	B	A
8	A	B	B	B
9	B	A	A	A
10	B	A	A	B
11	B	A	B	A
12	B	A	B	B
13	B	B	A	A
14	B	B	A	B
15	B	B	B	A
16	B	B	B	B

Color Elimination: Every time the player is able to eliminate a color as an option, the branching factor is reduced by one. For example, for the code (B B B B), guessing (A A A A) produces the response (0,0). According to the rules in Table 1, the player will remove A as an option. The search space is then reduced by:

$$|S| - (C-1)^P = 16 - (2-1)^4 = \mathbf{15}$$

possible solution. And, should we remove all solutions with an A in the space described by Table 3, we are indeed left with 1 possible solution, namely (B B B B).

Target Identification: Each time the player learns about new targets, we reduce the search space to:

$$|S| \leftarrow \binom{P}{\#} + (C - \#)^{P-\#}$$

where '#' is the number of targets the player identified. The intuition behind this is that now our player knows all future guesses should have some arrangement of the newfound targets ($= \binom{P}{\#}$) plus all possible solutions for a reduced problem size of (P-#)-pegs and (C-#)-colors. For example, guessing (A A A A) for the code (A B A B) produces the response (2,0). The player processes this response and identifies two new A targets. Therefore the new space now contains:

$$\binom{4}{2} + (2-2)^{4-2} = (6) + (0) = \mathbf{6}$$

possible solutions. Should we exclude all solutions with less than or more than 2 A's in the search space in Table 3, we are left with 6 valid guesses. It is interesting to note that in this specific example, no additional legal codes are supplied by the reduced search space $(C - \#)^{P-\#}$ since choosing where to place two As will trivially solve the other pegs for the second color. Finally, as more targets are identified, this evaluation is performed recursively over the reduced search space $(C - \#)^{P-\#}$ until it is empty.

3. Mystery SCSAs

In order to demystify the five mystery SCSAs, we performed a simple statistical analysis of the provided mystery codes. For each mystery code sample, we generated the general color distribution across all 100 codes, a color-peg heat map to explore peg color preference, and the distribution of codes with 1 to C colors. In addition, we produced the color distribution of codes with 1 to C colors.

Mystery-1 Sample: The distribution of codes with n colors (see Appendix) immediately reveals that all codes consist of three colors. A further examination of the codes in the sample reveals that those three alternate. For example

(E D C E D C E) (A D B A D B A) (E C A E C A E).

The color distribution across the entire sample suggests that all colors are equally likely to appear. It is therefore safe to assume that this SCSA implements a three-color-alternating strategy.

Mystery-2 Sample: Figure 1 (middle) shows that most codes have between 1 and 2 colors. The remaining codes (those with more than 2 colors) produce sample sizes that are too small to analyze. It is therefore reasonable to assume that this SCSA prefers fewer colors to many. The color distribution (top left) reveals a slight preference towards Cs, Es, and As in that order. Upon examinations of the n-color distributions, we see that when n = 1 and n = 2, the samples exhibit similar distributions, and both mirror the general color distribution across the entire sample.

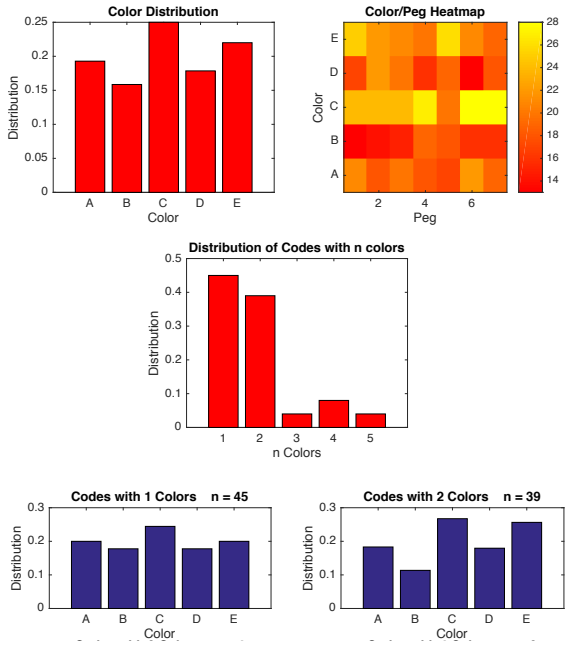


Figure 1. Mystery-2 Sample. Top left: color distribution across all 100 code samples. Top right: peg color preference heat map. Middle: distribution of codes with n colors as n goes from 1 to C. Bottom, color distributions of n=1 and n=2 respectively.

Mystery-3 Sample: Once again, the distribution of codes with n colors (see Appendix) reveals that all consist of three colors. The color distribution over the entire sample reveals a uniform distribution. In each code, the colors appear in the same quantity.

Mystery-4 Sample: The distribution of codes with n colors (see Appendix) shows that all codes consist of two colors. A careful examination of the sample codes reveals that the two colors alternate. For example

(C E C E C E C) (A B A B A B A) (D A D A D A D)

The color distribution across the entire sample suggests that all colors are equally likely to appear. It is

therefore safe to assume that this SCSA implements a two-color-alternating strategy.

Mystery-5 Sample: Figure 2 (middle) shows that most codes have between 3 and 4 colors. The remaining codes (those with less than 3 or more than 4 colors) produce sample sizes that are too small to analyze. It is therefore reasonable to assume that this SCSA prefers to incorporate 3 or 4 colors in its codes. The color distribution shows uniform distribution across colors.

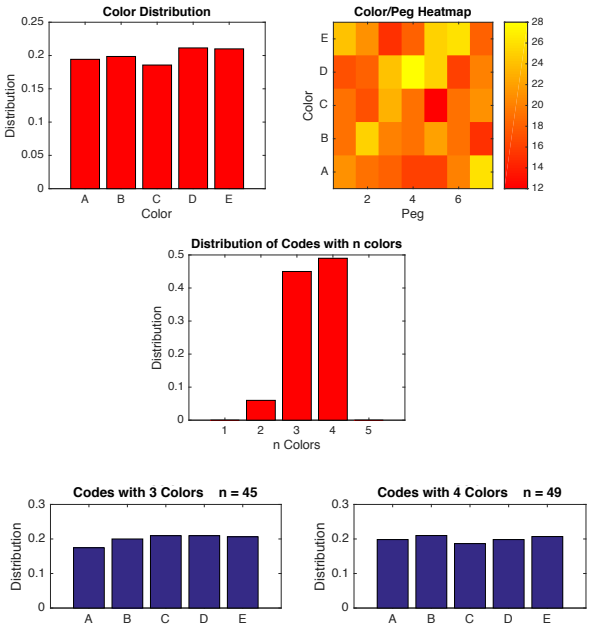


Figure 2. Mystery-5 Sample. Top left: color distribution across all 100 code samples. Top right: peg color preference heat map. Middle: distribution of codes with n colors as n goes from 1 to C. Bottom, color distributions of n=3 and n=4 respectively.

4. Strategy-Based Players

Due to time limitations, we only tested three different players, which incorporated different basic strategies, before finalizing brENIAC. All three players are based on and implement the approach described in subsection 1 of this section.

The first player chooses among all possible colors *in a* lexicographical order (A-Z), and therefore is called *Alphabetical* player (Alpha for short). For example, for a problem size (5,10) it will first guess (A A A A A), then (B B B B B), (C C C C C), etc. until the first target is found. Alpha will then continue to test colors from the filler queue *F* in a similar order. The example in Table 2 follows Alpha's solution.

The second player draws randomly from *F* and is therefore dubbed Random player (do not confuse with a fully random naïve player). For example, the Random player might start with (I I I I I), then attempt (E E E E E), (G G G G G), etc. until the first target is found. It will then continue to draw randomly from F to produce the next guess. Let's assume Random learned that there are two Es in the code. Then it will choose E from T and draw randomly, once again, from F, producing a guesses such as (E J J J J) or (E A A A A).

Finally, the third player, 2CA-Alpha, is based on Alpha player. However, a specialized strategy is applied when playing versus the 'two-color-alternating' SCSA. Given that there are only two colors and that they alternate, the player's main goal is to solve at least one peg as soon as possible. Once it does, locking every other peg with a similar color is trivial. The last part is to identify the second target (if it was not done so already during the process) and simply lock-in the remaining pegs with it.

As a baseline, we implemented an exhaustive player, which attempts to break the code using a brute-force approach. It does so by attempting every possible code begging with all As and finishing with all Zs for any problem size.

III. Results

In order to evaluate our players and determine which one performs better, we generated two sets of 20,000 single games. Each game was generated by randomly drawing a number between 4 and 26 for P-pegs and C-colors. However, for the first game set a SCSA was randomly drawn from a pool of eight different SCSAs for each game, while all games in the second set included the two-color-alternating SCSA. It is important to note that due to the nature of only-once, true random drawing of the number pegs and colors is not possible. Because every color can only appear once, the number of colors must be equal to or greater than the number of pegs ($C \geq P$). Therefore for only-once games, the number of colors was drawn first and was then used as a lower bound for the random drawing of the number of pegs. Figure 3 shows the distribution of games with problem size (P,C) as P and C go from 4 to 26, and the distribution of games by their SCSA strategy for the first set.

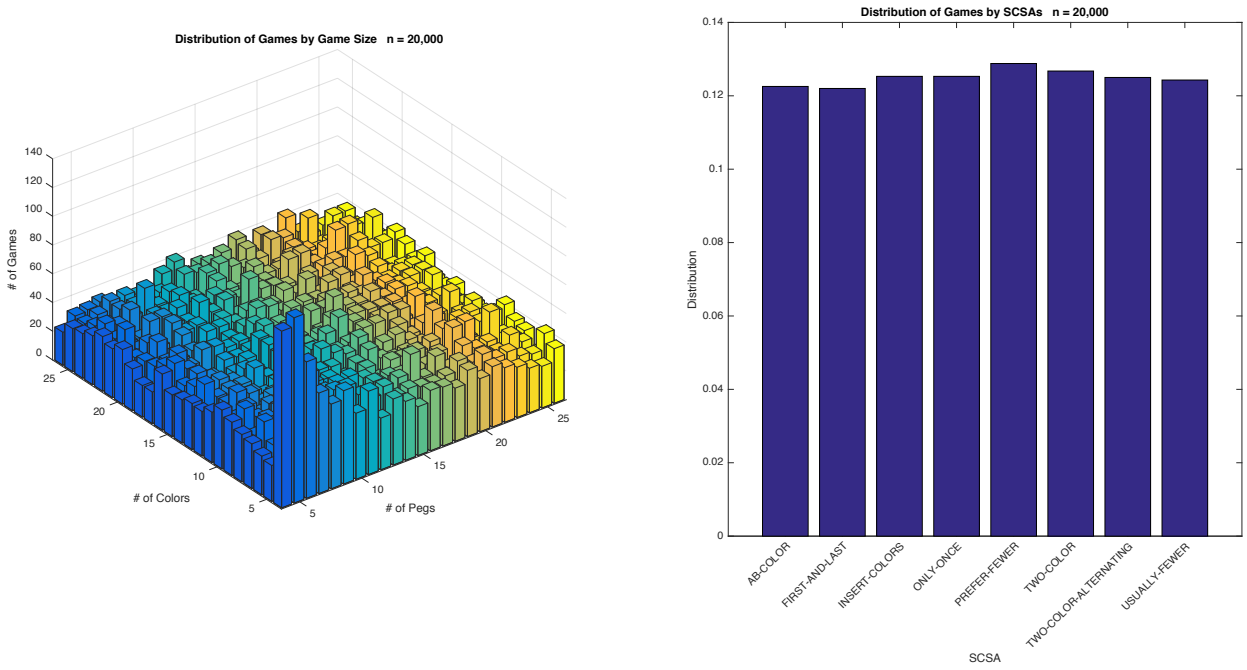


Figure 3. Distribution of games by problem size (left) as *P* and *C* go from 4 to 26, and the distribution of games by their SCSA strategy (right) for the first game set.

We first compared the performance between the exhastive player, Alpha, and Random. In order to evaluate the players' performance, we ran the three players over game set one (n = 20,000; problem sizes between 4,4 and 26,26; 8 different SCSAs) and recorded their individual score, number of guesses, and CPU time per game. These values were then averaged across each possible game size combination and plotted as a heat maps (Figure 4) to present the player's ability to scale in score, number of guesses, and CPU usage as the problems increase in size and complexity. Figure 4 shows a clear difference between the exhaustive player and Alpha and Random. The average score of the exhaustive player remains in the low 0 to 0.5 across all game sizes, while Alpha and Random both scale well between 2.4 to approximately 0.77 as the game size increase from (4,4) to (26,26). In addition, the same behavior is exhibited in the average number of guesses per game. Across all game sizes, the exhaustive either player meets the guess limit (average of 100 guesses) or uses 90 to 100 guesses on average. On the other hand, both Alpha and Random do not exceed approximately 60 guesses on average on the largest problem size (26,26). Finally it is evident that exhaustive player uses increasingly more CPU time as the problem size increases (up to 1600 on average), while Alpha and Random do not exceed an average of 1000 CPU time on the most difficult problem size.

In addition to the heat maps, we performed a two-sample t-test to decide whether the different players exhibit any significant difference in performance. When the exhaustive player's performance across each of the different SCSAs was compared to that of Alpha's and Random's, we found significant difference (wit 95% confidence) all across (Table 4). Similar differences were found when the number of guesses and CPU times were compared. However, the only significant difference between Alpha and Random is found over games of type ab-color, in which Alpha's performance is better ($\mu = 1.4891$; $p = 9.2541 \times 10^{-100}$). This makes perfect sense, since Alphabetical will first try guesses with As followed byt guesses with Bs, While Random might explore, in the worst case, 24 other colors before trying an A or a B. Similar differences found when average number of guesses and CPU time are compared (see Appendix).

Given these results, we decided that Alpha is the best player among the three and therefore decided to use Alpha as the base for the two-color-alternating specialized. Thus implementing 2AS-Alpha. Since the only thing that will differentiate between the performance of Alpha and 2CA-Alpha are games of type two-color-alternating, we used game set two to test the performance of the two players. As before, we produced heat maps to show scaling, and performed a two-sample t-test to find any significant difference.

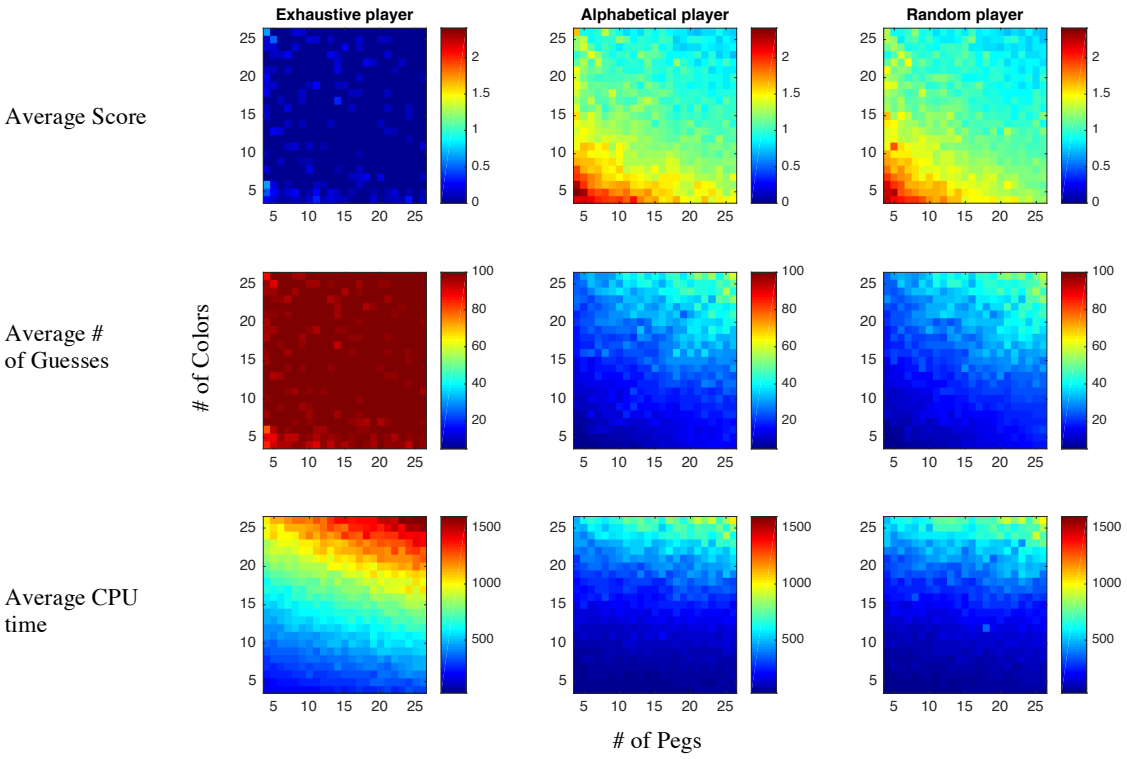


Figure 4. Distributions of average Score (top), number of guesses (mid), and CPU time (bottom) of the three players for scaling comparison.

There are clear differences in the heat maps produced by the two players over games of type two-color-alternating. It is evident that 2CA-Alpha is dominating in average score performance, average number of guesses required, and CPU time. In addition, the two-sample t-test showed significant differences across all three measurements. 2CA-Alpha has a better mean score (μ = 1.6927 vs. Alpha's μ = 1.1789 with; p = 0), a lower average number of guesses (μ = 10.8989 vs. μ =20.4107; p = 0) and requires less CPU time (μ = 80 vs. μ =239.1581). It is evident from these analyses that 2CA-Alpha is a better player than Alpha.

IV. Conclusion

Given the results presented in the previous section, we implemented brENIAC as the 2CA-Alpha player. It showed dominance over a naïve exhaustive player, and our Random player. In addition, it performed significantly better on games of type two-color-alternating, and therefore should perform better on longer tournaments.

Table 4. Two-sample t-test comparison between exhaustive player and Alpha (top) and Random (bottom).

SCSA	Exhaustive's Mean score	Alpha's Mean score	p value
'ab-color'	0.070757959	1.489059638	0
'first-and-last'	0.000819672	0.940066325	0
'insert-colors'	0.00054757	0.95290231	0
'only-once'	0.01501496	1.230281288	0
'prefer-fewer'	0.221539136	1.714675911	0
'two-color'	0.009252716	1.206807309	0
'two-color-alternating'	0.00089319	1.180265068	0
'usually-fewer'	0.006257031	1.161869351	0

SCSA	Exhaustive's Mean score	Random's Mean score	p value
'ab-color'	0.070757959	1.207071163	0
'first-and-last'	0.000819672	0.941736726	0
'insert-colors'	0.00054757	0.959830256	0
'only-once'	0.01501496	1.229963022	0
'prefer-fewer'	0.221539136	1.71682667	0
'two-color'	0.009252716	1.204689789	0
'two-color-alternating'	0.00089319	1.179950057	0
'usually-fewer'	0.006257031	1.161596711	0

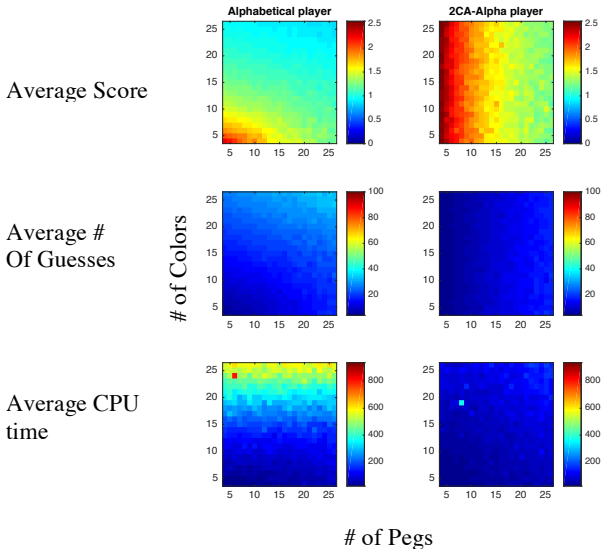


Figure 5. Distributions of average Score (top), number of guesses (mid), and CPU time (bottom) of Alpha and 2CA-Alpha for scaling comparison.

V. Division of Labor

All team members took an active part in the research, design, and development of the brENIAC player presented in this paper, or more specifically, the design of the inner state representation and the six rules of response processing.

Gil Dekel: Data and statistical analysis, figures, tables, and second to final paper draft.

Sarah-Ann Mathew: Research material organization, suggested to look closer into Rao's approach, and 50% of coding and debugging of brENIAC. Initial analysis of the mystery samples.

Ahmed Shah: Materials and references collection and organization, meetings note taking, and paper's first draft.

Jaspal Singh: 50% of coding and debugging of brENIAC. Testing and experimentation. Data collection and organization.

VI. References

Berghman, L., Goossens, D., & Leus, R. (2009). Efficient solutions for Mastermind using genetic algorithms. *Computers & operations research*, *36*(6), 1880-1885.

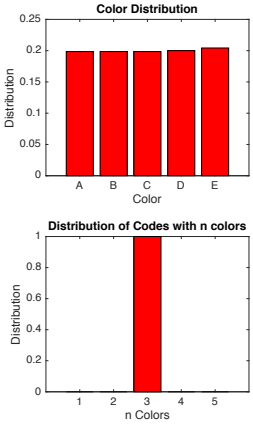
Knuth, D. E. (1976). The computer as master mind. *Journal of Recreational Mathematics*, *9*(1), 1-6.

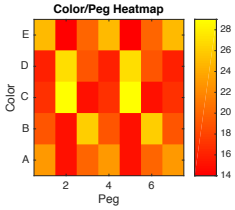
Merelo, J. J., Mora, A. M., Cotta, C., & Fernández-Leiva, A. J. (2013, January). Finding an evolutionary solution to the game of MasterMind with good scaling behavior. In *International Conference on Learning and Intelligent Optimization* (pp. 288-293). Springer Berlin Heidelberg.

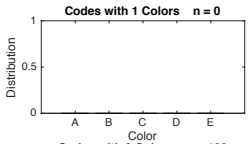
Rao, T. M. (1982). An algorithm to play the game of mastermind. *ACM SIGART Bulletin*, (82), 19-23.

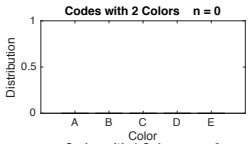
Appendix

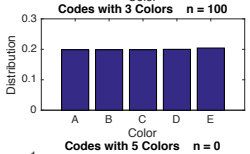
Mystery-1 Figures:

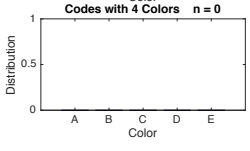


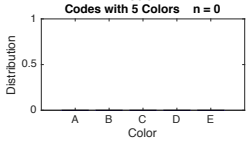




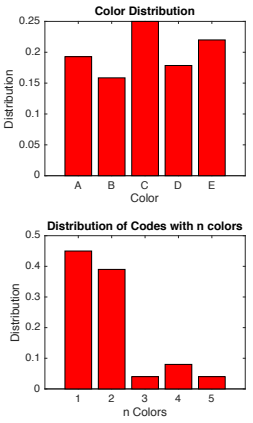


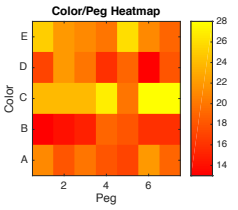


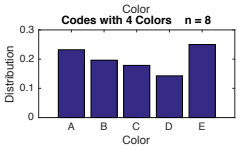




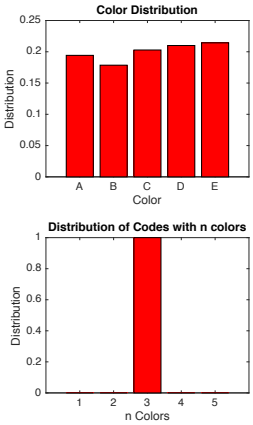
Mystery-2 Figures:

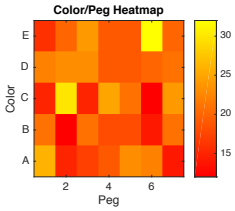


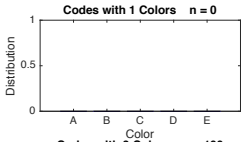


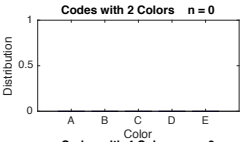


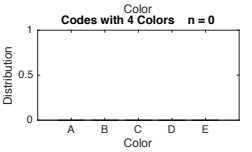
Mystery-3 Figures:

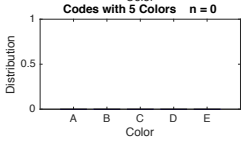




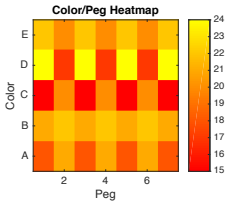


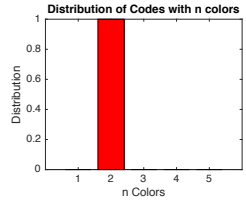


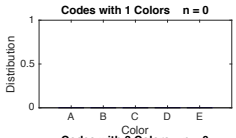


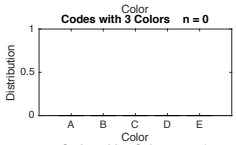


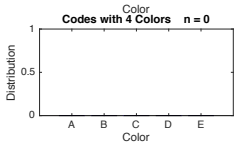
Mystery-4 Figures:

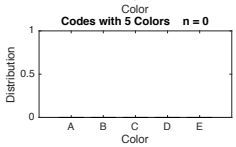




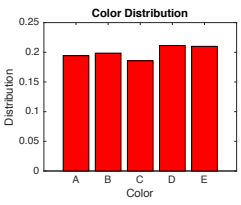


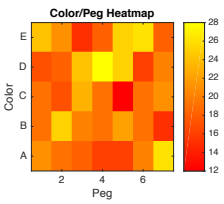


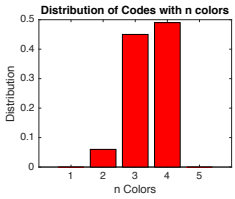


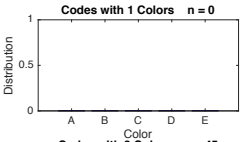


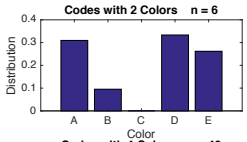
Mystery-5 Figures:

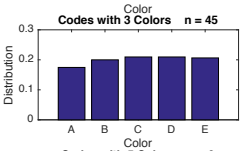


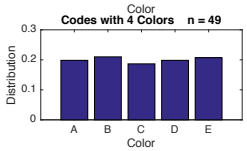


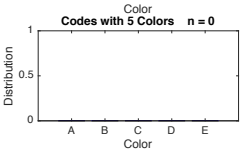












Statistical analysis for Exhaustive vs. Alpha vs. Random

Average scores:

SCSA	Exhaustive's Mean score	Alpha's Mean score	p value
'ab-color'	0.070757959	**1.489059638**	0
'first-and-last'	0.000819672	**0.940066325**	0
'insert-colors'	0.00054757	**0.95290231**	0
'only-once'	0.01501496	**1.230281288**	0
'prefer-fewer'	0.221539136	**1.714675911**	0
'two-color'	0.009252716	**1.206807309**	0
'two-color-alternating'	0.00089319	**1.180265068**	0
'usually-fewer'	0.006257031	**1.161869351**	0

SCSA	Exhaustive's Mean score	Random's Mean score	p value
'ab-color'	0.070757959	**1.207071163**	0
'first-and-last'	0.000819672	**0.941736726**	0
'insert-colors'	0.00054757	**0.959830256**	0
'only-once'	0.01501496	**1.229963022**	0
'prefer-fewer'	0.221539136	**1.71682667**	0
'two-color'	0.009252716	**1.204689789**	0
'two-color-alternating'	0.00089319	**1.179950057**	0
'usually-fewer'	0.006257031	**1.161596711**	0

SCSA	Alphas's Mean score	Random's Mean score	p value
'ab-color'	**1.489059638**	1.207071163	9.25E-100
'first-and-last'	0.940066325	0.941736726	0.861606681
'insert-colors'	0.95290231	0.959830256	0.474747116
'only-once'	1.230281288	1.229963022	0.98095594
'prefer-fewer'	1.714675911	1.71682667	0.937374982
'two-color'	1.206807309	1.204689789	0.810494738
'two-color-alternating'	1.180265068	1.179950057	0.966756701
'usually-fewer'	1.161869351	1.161596711	0.976157087

Average number of guesses:

SCSA	Exhaustive's Avg # of Guesses	Alpha's Avg # of Guesses	p value
'ab-color'	97.6303549571604	**14.4977560179519**	0
'first-and-last'	99.9385245901639	**36.8418032786885**	0
'insert-colors'	99.9632881085395	**36.1851556264964**	0
'only-once'	99.1496408619314	**25.1049481245012**	0
'prefer-fewer'	95.5512422360249	**15.2457298136646**	0
'two-color'	99.6919132149901	**19.8611439842209**	0
'two-color-alternating'	99.9416000000000	**20.3016000000000**	0
'usually-fewer'	99.7216411906677	**22.1130329847144**	0

SCSA	Exhaustive's Avg # of Guesses	Random's Avg # of Guesses	p value
'ab-color'	97.6303549571604	**19.9404324765402**	0
'first-and-last'	99.9385245901639	**36.6577868852459**	0
'insert-colors'	99.9632881085395	**35.7206703910615**	0
'only-once'	99.1496408619314	**24.9648842777334**	0
'prefer-fewer'	95.5512422360249	**15.1537267080745**	0
'two-color'	99.6919132149901	**19.8725838264300**	0
'two-color-alternating'	99.9416000000000	**20.3520000000000**	0
'usually-fewer'	99.7216411906677	**22.0985518905873**	0

SCSA	Alphas's Mean score	Random's Avg # of Guesses	p value
'ab-color'	**14.4977560179519**	19.9404324765402	5.82882975116352e-142
'first-and-last'	36.8418032786885	36.6577868852459	0.761204819413352
'insert-colors'	36.1851556264964	35.7206703910615	0.438167010230588
'only-once'	25.1049481245012	24.9648842777334	0.807505234615086
'prefer-fewer'	15.2457298136646	15.1537267080745	0.773898863390379
'two-color'	19.8611439842209	19.8725838264300	0.957441911952364
'two-color-alternating'	20.3016000000000	20.3520000000000	0.813173589858395
'usually-fewer'	22.1130329847144	22.0985518905873	0.963905154618314

Average CPU time:

SCSA	Exhaustive's Avg CPU time	Alpha's Avg CPU time	p value
'ab-color'	743.963688290494	**225.062015503876**	0
'first-and-last'	755.235245901639	**421.499590163934**	1.05444951931901e-182
'insert-colors'	732.886671987231	**410.768555466880**	1.00271609901093e-178
'only-once'	482.521947326417	**203.547885075818**	1.81275338004322e-225
'prefer-fewer'	736.802018633540	**146.157608695652**	0
'two-color'	752.291913214990	**228.511637080868**	0
'two-color-alternating'	734.976800000000	**231.426000000000**	0
'usually-fewer'	731.963395012068	**244.144006436042**	0

SCSA	Exhaustive's Avg CPU time	Random's Avg CPU time	p value
'ab-color'	743.963688290494	**229.718074255406**	0
'first-and-last'	755.235245901639	**436.297950819672**	7.56447205815831e-155
'insert-colors'	732.886671987231	**419.451316839585**	5.57359306658699e-173
'only-once'	482.521947326417	**213.337190742219**	1.73068129844688e-201
'prefer-fewer'	736.802018633540	**154.165372670807**	0
'two-color'	752.291913214990	**239.872978303748**	0
'two-color-alternating'	734.976800000000	**239.609200000000**	0
'usually-fewer'	731.963395012068	**255.750201126307**	0

SCSA	Alphas's Avg CPU time	Random's Avg CPU time	p value
'ab-color'	225.062015503876	229.718074255406	0.392813419060326
'first-and-last'	421.499590163934	436.297950819672	0.198320596386499
'insert-colors'	410.768555466880	419.451316839585	0.418520876649322
'only-once'	203.547885075818	213.337190742219	0.219657662146024
'prefer-fewer'	146.157608695652	154.165372670807	0.141051515630041
'two-color'	**228.511637080868**	239.872978303748	0.0357725563420370
'two-color-alternating'	231.426000000000	239.609200000000	0.133009056935359
'usually-fewer'	244.144006436042	255.750201126307	0.0639300338335033