DOI: 10.2478/ijasitels-2020-0003

On Hagelbarger's and Shannon's matching pennies playing machines

Macarie BREAZU[1], Daniel VOLOVICI[1],
Daniel I. MORARIU[1], Radu G. CREȚULESCU[1]
[1]Computer Science and Electrical and Electronics Engineering Department, Faculty of Engineering, "Lucian Blaga" University of Sibiu, Romania
{macarie.breazu, daniel.volovici, daniel.morariu, radu.kretzulescu}@ulbsibiu.ro

Abstract
In the 1950s, Hagelbarger's Sequence Extrapolating Robot (SEER) and Shannon's Mind-Reading Machine (MRM) were the state-of-the-art research results in playing the well-known "matching pennies" game. In our research we perform a software implementation for both machines in order to test the common statement that MRM, even simpler, beats SEER. Also, we propose a simple contextual predictor (SCP) and use it to compete with SEER and MRM. As expected, experimental results proves the claimed MRM superiority over SEER and even the SCP's superiority over both SEER and MRM. At the end, we draw some conclusions and propose further research ideas, like the use of mixing models methods and the use of Hidden Markov Model for modelling player's behaviour.

Keywords: matching pennies, Hagelbarger, sequence extrapolating robot, Shannon, mind reading machine, contextual predictor

1. Introduction. Game description

One of the simplest games that can be played by two opponents is the "matching pennies" game [6]. Both players choose a face of their own coin ("head" or "tail") in secret and, after that, both expose their choices. One player wins if the choices are the same (hence the name "matching pennies"), the other when the choices are different. Even if theory says that, at random play, you neither win nor lose on long term, people's play falls in patterns and this can be used by the opponent. The common people's idea is that your smartness is what wins the game. Edgar Allan Poe's character from "The Purloined Letter" uses also body-language signs to predict opponents moves at this game.

Building a machine that plays this game requires the two fundamental steps in machine learning: building a model of the problem's world (in this case of the opponent) and using this model to implement the behaviour. Therefore, being

a redoubtable problem from the (today called) machine learning point of view, it got research focus from the beginning of the artificial intelligence era.

In the 1950', at Bell Laboratories, David Hagelbarger and Claude Shannon each build their own machines to play the matching pennies game with lab colleagues (Claude Shannon built also other interesting machines [7]). For today such machines can be considered trivial, but for that moment, when the memory was implemented with relays, these were state-of-the-art research results, and therefore the machines got their remarkable place in the history of computer science.

Hagelbarger calls his machine "SEquence Extrapolating Robot' (SEER) [1], focusing on the feature that extrapolates the series of the opponent's play. Shannon calls his machine "Mind Reading machine" (MRM) [2], focusing on the feature that it models the opponents' mind. The Shannon's MRM machine was a simplified version of the Hagelbarger's SEER machine, but even so it hoped for and got better results. Images with the SEER and MRM machines can be found in [8].

Different from the SEER and MRM papers, where "+" and "–" symbols were used to describe the play, we will describe the play using the "H" (Head) and "T" (Tail) symbols (according to the "matching pennies" game's name).

An example of a matching pennies game evolution can be:

#play	Player 1 (plays for matching)	Player 2 (plays for different)
1	T	H
2	H	H
3	T	T
4	T	H
5	H	H
6	H	T
7	H	T
8	…	…

Win/Loss (for player 2)	Change (for player 2)
1 (Win)	
0 (Loss)	0 (Same)
0 (Loss)	1 (Different)
1 (Win)	1 (Different)
0 (Loss)	0 (Same)
1 (Win)	1 (Different)
1 (Loss)	0 (Same)
…	…

In both Hagelbarger and Shannon machines, the machines were designed to play for "matching the pennies", i.e. to read the opponent's mind (hence the name "mind-reading machine" used by Shannon).

2. Hagelbarger's SEER

The Hagelbarger's SEER (SEquence Extrapolating Robot) is presented in [1]. Its following description is taken/adapted from that paper; all quotation marks refer to it. The block diagram of SEER is presented in Fig. 1.

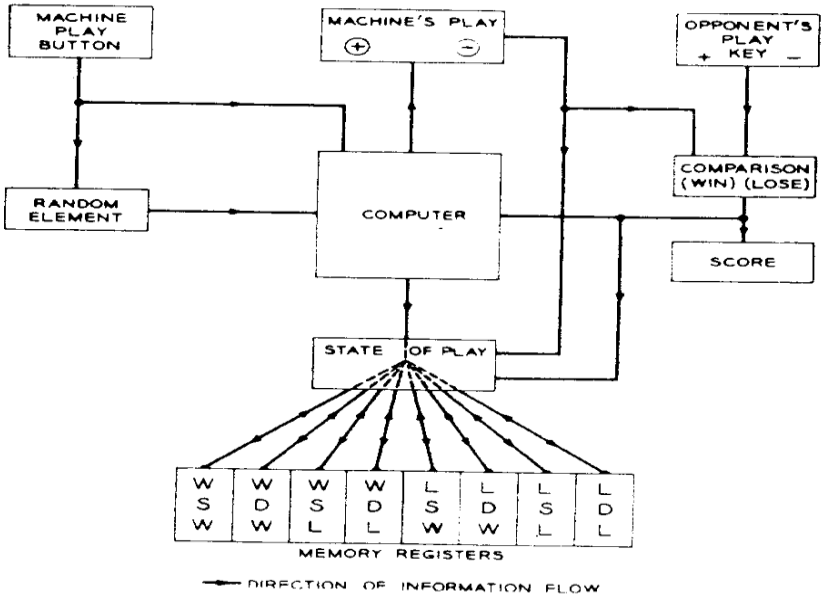

Figure 1. Hagelbarger's SEER block diagram [1]

"The "state of play" of the machine is determined by three things:" (presented in the order they appear in time)

- "whether it won or lost play before last play" (labelled W/L)
- "whether it played same or different last time" (labelled S/D)
- "whether it won or lost last play" (labelled W/L)

Based on those three bits of information (W or L, S or D, W or L) we have 8 different states of play, labelled in Fig. 1 from WSW to LDL.

For each state the machine's memory "stores two kinds of information:

a) Should the machine play same or different in this state in order to win?
b) Has the machine been winning in this state?"

"The a) part of the memory state is controlled by a reversible counter which starts at zero and can count up to +3 and down to -3. At the end of each play, if the machine should have played same, one is added to the counter. If it should have played different, one is subtracted. The counter will thus contain the number of times the machine should have played same in that state minus the number of times it should have played different. The stops at +3 and -3 in effect make the machine forget ancient history."

The b) part is implemented by "remembering whether the machine has won both, one, or neither of the last two plays in that state."

The machine's play is determined by the following rules:

"If the machine has lost the last two times in the present state, it plays randomly with equal odds on *same* and *different*.

If the machine has won one of the last two times in this state, it has three-to-one odds that it will follow the instruction in the a) part of the state memory.

If the machine has won both of the last two times in this state, the machine must follow the instruction in the a) part of the state memory."

After each play the machine updates the content of the state memory and the "state of play" and is ready for the next play.

For testing it, we have simulated the SEER machine in C++ language. An example of running our SEER implementation program (here against another SEER machine) is given in the following:

Game evolution (after 27 plays):

```
ME: HTTHHTTTTHHTHHHHHTTTTTTHTHT (SEER machine)
HE: TTHHTHHTTHTHHTHHHHHHHHHHTHH (opponent)
WH: 01010001110010111000000111 10 (win history - for machine)
CH:  10101000101100001000001111 1 (change history - for machine)
```

Machine's state at the end of the sequence:

```
Counts: -3  2 -2 -3  2  1 -1 -1 (the 0-7 state counters)
WinHR:   1  3  1  1  1  2  0  2 (the 0-7 state win history registers)
StateOfPlay: 6      (given by the gray 110 – WinOld_Change_WinNew)
```

In the previous example the win history register contains an integer value built from the 2 last win bits (previous win is MSB, last win is LSB). Because the win history register for `StateOfPlay=6` is 0, the machine will play randomly the next play.

3. Shannon's Mind-Reading Machine

The Shannon's Mind-Reading Machine (MRM) is described in [2]. As expected the machine looks a lot like SEER. The main difference is the fact that it tries to model the opponent's behavior; therefore the wins and changes are computed from the point of view of the opponent.

Like in SEER, MRM considers the 8 situations (states) based on the history from WSW to LDL (same labels as in SEER description). In each situation the player can do two things: "he may then play the same or differently". If, in one state, the same action of the opponent (play same or differently) appears in both the last 2 times, this behavior is considered to be repeated by the opponent's play and is used for the play of the machine. If this repetition of the opponent's play does not appear, the machine plays randomly.

From the description above we can notice that the machine does not use the StateOfPlay-based 7 state counters (hard to implement by relays) anymore, it uses only the StateOfPlay-based 2-bit change history register. This is in line with the statement made in [1]: "C.E. Shannon has built a machine using about half as many relays which follows a simplified version of the same strategy."

For testing it, we have simulated the MRM machine in C++ language. An example of running our MRM implementation program (here against another MRM machine) is given in the following:

Game evolution (after 27 plays):

```
ME: TTTTHTHHTTTHHHHHTTHHHTHTTTH (MRM machine)
HE: HHHHHHTTHTTTHTTHTTTTHTTTHTT (opponent)
WH: 11110111100101100011001010 1 (win history - for opponent)
CH:  0000010110011011000110011 0 (change history – for opponent)

ChgHR:  0  0  0  1  1  3  2  1 (the 0-7 state change history registers)
StateOfPlay: 1          (given by the gray 001 – WinOld_Change_WinNew)
```

In the previous example the win change register contains an integer value built from the last 2 change bits (previous change is MSB, last change is LSB). Because the change history register for StateOfPlay=1 is 0, the machine will assume the opponent will not change and therefore opponent will play same as previous – so that the machine will play same as the opponent, i.e. 'T'.

Obvious, in our C++ language implementations of SEER and MRM, we do not have the restrictions of the relay-based implementations of Hagelbarger and Shannon - i.e. we can maintain the whole histories for understanding/ debugging reasons – and also we can easily have a random number generator (the C library function `random`).

4. Experimental results (1) – SEER vs MRM

In [1] it is stated that "After much discussion an umpire machine was built which connected the two machines, and they were allowed to play several thousand games. The agility of the small machine triumphed, and it beat the larger one about 55-45."

In order to verify that statement, we made some tests in which the two machines play one against the other. Both machines are playing "same" (try to "match pennies"), it is the responsibility of the embedding environment to present to one of the machines the opposite selection of the other.

Because it is not clear from papers what was the goal of the machines: (1) to be the first winning a number of plays or (2) to achieve most wins from a fixed number of plays, we present the results for approach (2). This allows us to verify if the winner is the same after short or long games, therefore we present the results after 50, 100 and 200 plays for each game (winner in bold).

Table 1. SEER vs. MRM results

Game number	Results after 50 plays	Results after 100 plays	Results after 200 plays
Game 1	**26**-24	47-**53**	89-**111**
Game 2	18-**32**	38-**62**	77-**123**
Game 3	24-**26**	38-**62**	83-**117**
Game 4	23-**27**	49-**51**	88-**112**
Game 5	20-**30**	42-**58**	87-**113**
Game 6	**27**-23	48-**52**	95-**105**
Game 7	20-**30**	40-**60**	83-**117**
Game 8	22-**28**	45-**55**	88-**112**
Game 9	**27**-23	49-**51**	91-**109**
Game 10	24-**26**	46-**54**	91-**109**
Average	23.1 – **26.9**	44.2-**55.8**	87.2-**112.8**

We can notice from the results that, for the beginning of the games, SEER has some chances against MRM but, in long enough games, loses in all cases. The result can be explained by the fact that, at the beginning, state memory does not contain enough data for good predictions, so the results tend to be random.

In Fig. 2 we present the step-by-step evolution for the first 100 plays of Game 1. We can observe that at the beginning SEER has some advantage but MRM recovers and, in the end, it wins.

Even if it is not obvious what "the small machine … beat the larger one about 55-45" means, our average result (for 100 play games) of 55.8-44.2 in favour of MRM looks remarkable. The "55-45" statement suggests that the original games consisted of 100 plays. This is why we have chosen to present in figures the first 100 plays only.

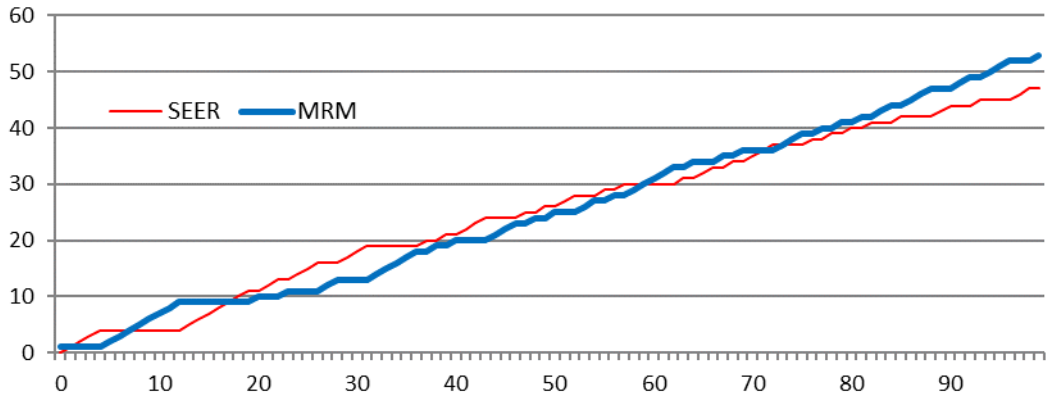

Figure 2. Score evolution for SEER vs. MRM Game 2 (first 100 plays)

5. A simple contextual predictor

It is obvious that both SEER and MRM suffer from the (hardware) limitation of history length that can be part of the stateOfPlay definition. Therefore we propose and test a simple contextual predictor (SCP), where such a limitation is overcome. The main idea is to find repeated patterns in opponents' play and suppose that he will repeat the same pattern over again.

In order to simplify the description of the algorithm and the implementation debugging we build a history string containing the description of each play (for the opponent's point of view) using the following symbols:

- `sw` = play same and win
- `sl` = play same and lose
- `dw` = play different and win
- `dl` = play different and lose

For a game evolution (after 33 plays):

```
ME: THHHHTHHTHHHHHTHHHTHHTTTHTHTHTTTT (SCP machine)
HE: HHTHHHHHTTHHHTTHTTHHHHHTHHHTTHHTT (opponent)
WH: 10100100010001001110011001001110O (win history)
CH:  011000010100101101000011001010O (change history)
```

(the win and change history are from the opponent's point of view)

we get the history string:

```
sldwdlslswslsldlswdlslsldwsldldwswdwslslswswdldlswsldlswdwswdlsl
```

We define the **context of length N** as **the last N** (**double) symbols** from the string. Because of the semantics of the build history string we search for matches only with a step of 2 individual symbols.

We start by searching in the history string the occurrences of the context string for the context length N=1. For each match we consider the following `sw,sl,dw` or `dl` symbol and increment its counter. Then we increase the context length and repeat the procedure until we get 0 or 1 total occurrences of the current context in the history string.

We then predict the next move of the opponent based on the longest context where the values for **`dl+sw`** (when we expect he will play *same*) and **`sl+dw`** (when we expect he will play *different*) are unbalanced (not equal). Certainly, if we have no match even at length 1 or, in all the analyzed contexts, the results are balanced we use a random play.

For the previous game example we have:

First step: length of context=1, context="`sl`"

```
sldwdlslswslsldlswdlslsldwsldldwswdwslslswswdldlswsldlswdwswdlsl
```

#next occurrences: `dl=3,sw=2,sl=3,dw=2, dl+sw=`**`5`**`, sl+dw=`**`5`**

Second step: length of context=2, context="dl

sldwdlslswslsldlswdlslsldwsldldwswdwslslswswdldlswsldlswdwswdlsl

#next occurrences: dl=0,sw=1,sl=1,dw=0, dl+kw=**1**, sl+dw=**1**

Third step: length of context=3, context="swdl

sldwdlslswslsldlswdlslsldwsldldwswdwslslswswdldlswsldlswdwswdlsl

#next occurrences: dl=0,sw=0,sl=1,dw=0, dl+sw=**0**, sl+dw=**1**

Because the total number of occurrences is 1, we do not further increase the context length and stop the analysis.

The longest unbalanced context is the one for length 3, so we predict based on it. The opponent will then choose to play **d** (different) relative to its last play '**T**', so the SCP predictor will predict also '**H**' (the SCP plays for "matching pennies").

The algorithm was described in "context" language, commonly used in data compression. In other research areas it is called "prediction by partial matching" (e.g. in [5]).

6. Experimental results (2) – SCP vs. SEER and vs. MRM

We also test our proposed SCP against SEER and against MRM, in the same approach like previous, i.e. in 10 games evaluated after 50, 100 and 200 plays. Table 2 and Fig. 3 present results of SCP against SEER while Table 3 and Fig. 4 present results of SCP against MRM.

Experimental results prove that SCP beats both SEER and MRM in long games. After 50 plays SEER and MRM win in few cases, but for longer games SCP recovers always.

Table 2. SCP vs. SEER results

Game number	Results after 50 plays	Results after 100 plays	Results after 200 plays
Game 1	**30**-20	**56**-44	**116**-84
Game 2	24-**26**	**53**-47	**111**-89
Game 3	25-25	48-**52**	**106**-94
Game 4	**30**-20	**63**-37	**124**-76
Game 5	**27**-24	50-50	**108**-92
Game 6	25-25	**53**-47	**105**-95
Game 7	**33**-17	**60**-40	**119**-81
Game 8	25-25	**55**-45	**110**-90
Game 9	24-**26**	**53**-47	**114**-86
Game 10	**29**-21	**55**-45	**113**-87
Average	**27.2**-22.9	**54.6**-45.4	**112.6**-87.4

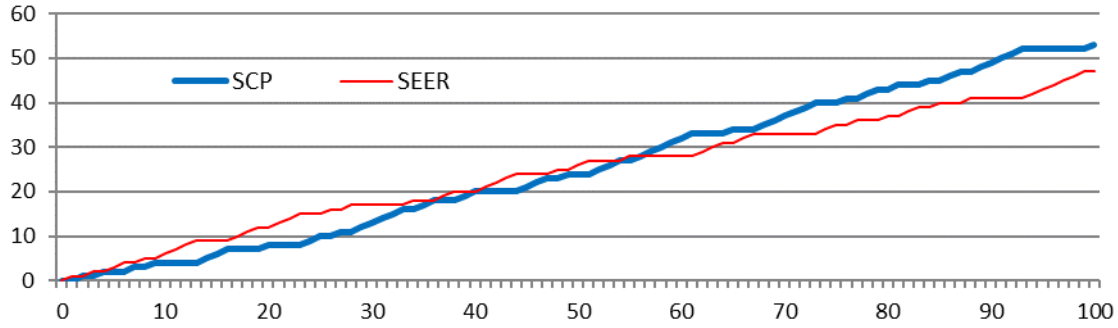

Figure 3. Score evolution for SCP vs. SEER Game 2 (first 100 plays)

Table 3. SCP vs. MRM results

Game number	Results after 50 plays	Results after 100 plays	Results after 200 plays
Game 1	**33**-17	**61**-39	**121**-79
Game 2	**31**-19	**61**-39	**115**-85
Game 3	23-**27**	**53**-47	**110**-90
Game 4	**33**-17	**65**-35	**120**-80
Game 5	**27**-23	**53**-47	**102**-98
Game 6	**35**-15	**61**-39	**117**-83
Game 7	**30**-20	**60**-40	**113**-87
Game 8	**28**-22	**53**-47	**112**-88
Game 9	24-**26**	**53**-47	**110**-90
Game 10	22-**28**	**53**-47	**117**-83
Average	**28.6**-21.4	**57.3**-42.7	**113.7**-86.3

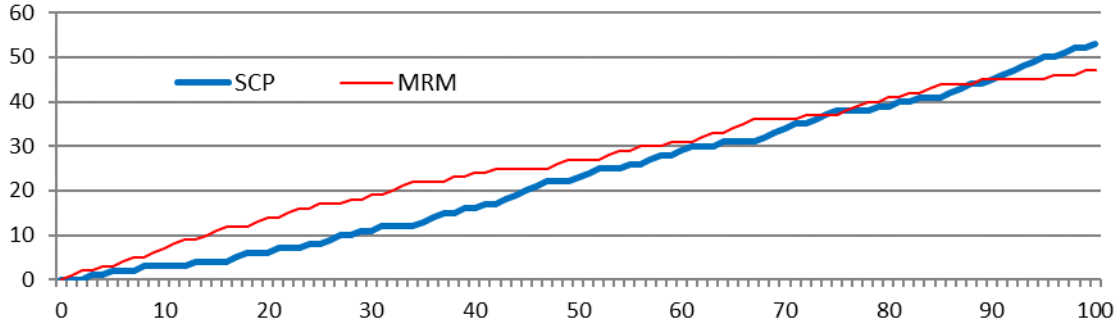

Figure 4. Score evolution for SCP vs. MRM Game 3 (first 100 plays)

The games selected to be presented in Fig. 3 and Fig. 4 are ones where, at the beginning, SCP loses, but the recovering process can be observed. The recovering at 100 plays is not by chance, it continues so that at 200 plays the win is consistent.

In order to evaluate what was the maximum length of context that was found during the play and how often it was used for prediction (i.e. it was not given up because it was balanced) we saved, for each prediction, the length of the longest context found (LLCF) and the length of the context used for prediction (LCUP). In table 4 we present the number of occurrences for each such pair, computed as sum for all the 10 SCP vs. MRM games analyzed.

Table 4. Number of contexts found and used

		Length of the context used for prediction (LCUP)						Total
		0	1	2	3	4	5	
Length of the longest context found (LLCF)	0	60						60
	1	48	86					134
	2	50	140	296				486
	3	32	82	183	638			935
	4	1	4	8	33	327		373
	5	0	0	0	0	0	12	12
Total		191	312	487	671	327	12	2000

From the previous table we notice that the maximum context length found and used (in 200 play games) is 5, and the longest context found is, in most cases, also used for prediction (corresponding to the numbers from the first diagonal).

In order to see how LCUP and LLCF evolve during play, we present in Fig. 5 their evolution during the first 100 plays of game 3 SCP vs. MRM (game also presented in Fig. 3). When the longest context found is also used for prediction the red diamond point is no more visible (it is overlapped by the blue square). As the game evolves longer contexts are found and used.

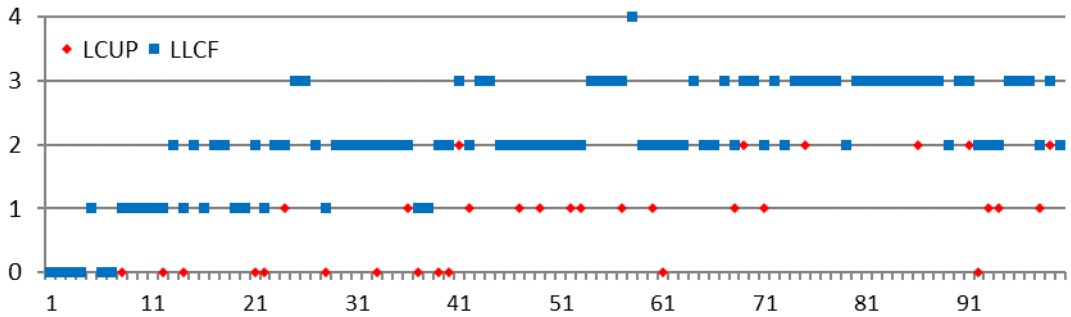

Figure 5. Evolution of LCUP and LLCF for SCP vs. MRM Game 3 (first 100 plays)

The SCP predictor was used also against some human players. We do not present such results because the results are very person dependent, and not statistically relevant. But, like with SEER and MRM tests done, the machine usually won. A rigorous set of tests for SCP (and maybe also for SEER and MRM) should be done in future, with enough games played so that the results became statistically relevant.

7. Conclusions and future work

The study of the SEER and MRM machines done in order to implement them in software reveals how remarkable was their design and implementation, considering the technological level at that time, i.e. implementing memory only by relays.

Our experimental results are consistent with the statement that MRM, although simpler, beats SEER. Also, the proposed SCP beats both SEER and MRM. This result is not surprising, previous hardware limitations being overcome

Another possibility that can be considered for SCP is to evaluate, in each context, the value **`sl+sw`** against **`dl+dw`** (that is, to consider the opponent less skilled, concerned only about changing or not in the current context). We assume that this strategy could be appropriate mainly for human opponents, but it needs to be tested.

In our implementation we have used for prediction always the longest unbalanced context. It could be also useful to use all the lower order unbalanced contexts (but weighted somehow differently). For context mixing we can use the approach from PAQ methods [3]. Unfortunately, because of the short length of the games, only simple contexts and mixing models can be considered.

The use of Hidden Markov Model HMM [4] looks promising for this application. After each play can we train the HMM model to learn the game history string (using a 4-symbol alphabet corresponding to the kw, kl, cw, cl cases) and therefore to model the behaviour of the opponent. After learning, the model is used to predict the next symbol and, from it, the opponent's move. After having the information of the next play the training and predicting process repeats for the entire game.

Our further research will focus on the idea of mixing different context length predictors and on the use of HMM for the matching pennies game.

References

[1] DW Hagelbarger. *Seer, A SEquence Extrapolating Robot*, IRE Transactions on Electronic Computers, page 1–7, March, 1956.
[2] Claude E Shannon. *A Mind-Reading(?) Machine*, Bell Laboratories Memorandum, March 18, 1953.
[3] Matthew V. Mahoney. *Adaptive Weighing of Context Models for Lossless Data Compression*, Florida Institute of Technology CS Dept. Technical Report CS-2005-16, 2005.
[4] Lawrence R. Rabiner, *A tutorial on HMM and selected applications in Speech Recognition*, Proceedings of the IEEE, vol 77, no. 2, 1989.
[5] Arpad Gellert, Adrian Florea. *Web prefetching through efficient prediction by partial matching*, World Wide Web, volume 19, pages 921–932, 2016.
[6] https://en.wikipedia.org/wiki/Matching_pennies
[7] https://cs.stanford.edu/people/eroberts/courses/soco/projects/1999-00/information-theory/ai.html
[8] http://william-poundstone.com/blog/2015/7/30/how-i-beat-the-mind-reading-machine