

ZX MASTER MIND



Master Mind. The board game.

The game, in the classic version, consists of a decoding board and two types of colored pegs:

- *Code pegs* of six different colors are used for the composition of the secret code and attempted codes.
- *Key Pegs* smaller than the first ones, are black and white and are used for the composition of the key codes.

(In the image above my set in the 8-color variant bought in the early 80's)

Game rules

The Master Mind is a logic game where two players alternate in the roles of codemaker and codebreaker.

The codemaker is the one who chooses and hides the Secret Code and the codebreaker is the player who must guess it.

Players alternate roles in each turn trying to guess the secret code in the least number of attempts.

The two players will decide at the beginning how many turns to make and at the end they will compare the sum of the attempts. Whoever has the lowest number is the winner.

The secret code is made up of four colored pegs kept hidden by a shield of the decoding board.

For subsequent attempts, the codebreaker will play codes, also made up of colored pegs, and the codemaker will place key pegs of black and white color for each attempt.

Based on the codemaker responses, the codebreaker will decide the next code to play.

The key codes in response to each attempt follow these rules:

- As many black pegs as there are colors of the attempt that are present in the secret code and that are also in the right position.
- As many white pegs as there are colors in attempt that are present in the secret code but in the wrong position.
- No key peg for the colors of the played code not present in the secret code
- Note: The white and black pegs indicate the number of colored pegs present in the secret code but not their position.

To avoid mistakes in case of repeated colors, it is preferable that the codemaker puts the black pegs first. Afterwards the white pegs will have to exclude the colors already considered in the answer of the black pegs. In the following example the codemaker puts only one black peg for blue present and in the right position. No white peg for the other blue. No other peg for the other colors as they are not present in the secret code.



Possible combinations of secret codes.

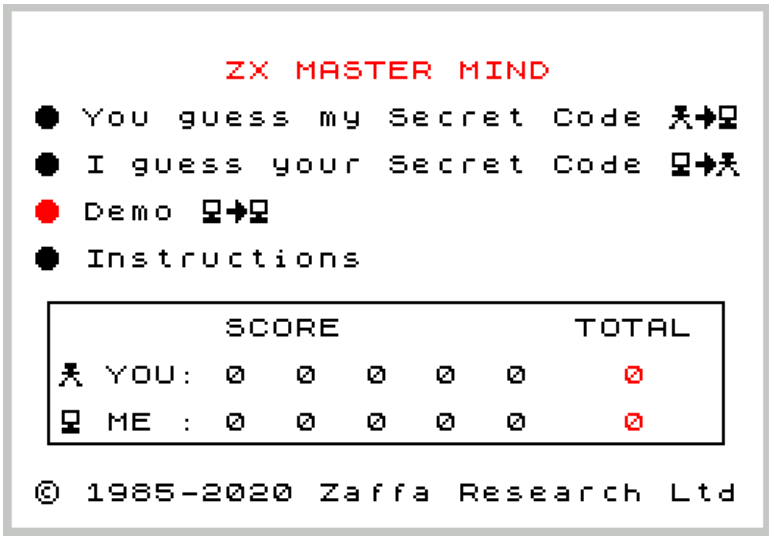
In the classic version, the total number of possible Secret Codes is calculated according to the formula of the variations with repetition of the combinatorial calculation: $VR_{n,k} = N^k$ where n is the number of colors and k the number of positions, then $6^4=1296$. For convenience, even if improperly, these variations will be called combinations. There are other variations of the game with more colors and positions or facilitated versions with less colors / positions for children. In the classic version the colors are white, blue, red, yellow, black, green. The 8-color variant adds orange and brown. Compared to the colors of the board game, the colors used by the program are those of the ZX Spectrum, that are: blue, red, magenta, green, cyan and yellow. Black and white colors are used only for key codes.

ZX Master Mind. The Master Mind Program for ZX Spectrum.

The following options are available from the main menu:

- The player guesses the secret code chosen randomly by the program
- The program guesses the secret code chosen by the player
- Demo mode - The program guesses a random secret code
- Instructions

Below the menu there is the score for the games played between the player and the program.



Here are the details for each option:

The player guesses the secret code.

The player is the codebreaker and the program is the codemaker.

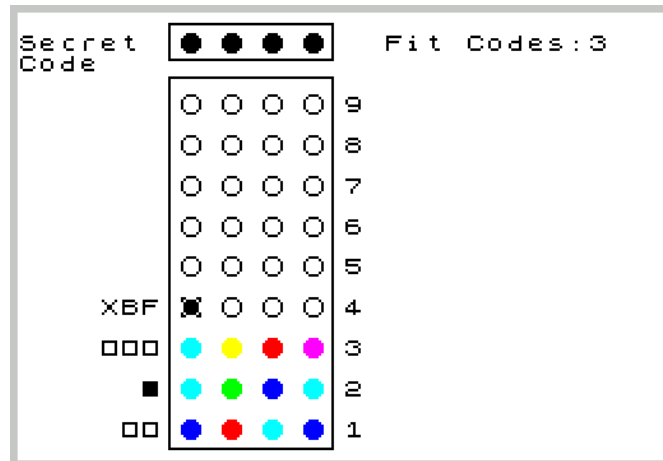
The program will choose a random secret code which it will keep obscured at the top of the decoding board.

After every attempt code played, the program will assign a key code (white and black pegs).

The "X" option exit from the game and come back to the menu.

For options "F" and "B", see paragraph: "Options F and B"

At the top right are displayed the "Fit Codes" that are the number of possible secret codes remaining after each attempt.

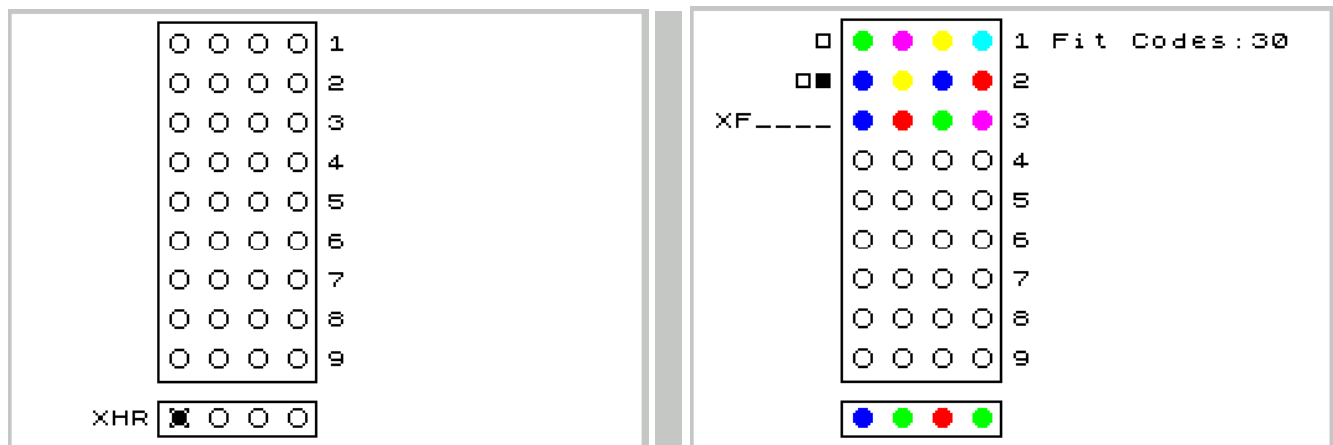


The program guesses the secret code.

The program is the codebreaker and the player is the codemaker.

At the start of the game, the player has three options for choosing the secret code:

- 1) Choose a color for each position (see instructions option for the use of keyboard and Joystick)
- 2) Use the "R" (Random) option to generate a random secret code
- 3) Use the "H" (Hidden) option to keep your secret code written down on a sheet.



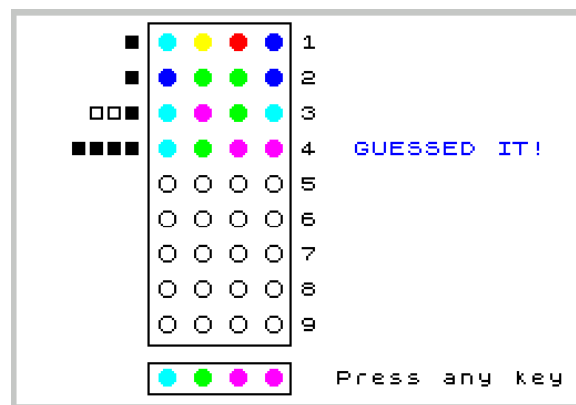
After entering the secret code, the program will start playing the first attempt to which you will have to provide a key code.

The "X" option exit from the game and come back to the menu.

The "F" option will be discussed in the paragraph: "Options F and B"

The program guesses a random secret code - Demo mode

The program will play the role of both Codemaker and Codebreaker. It will randomly choose a Secret Code and play until it guesses it. This mode can be used to understand how the game works.

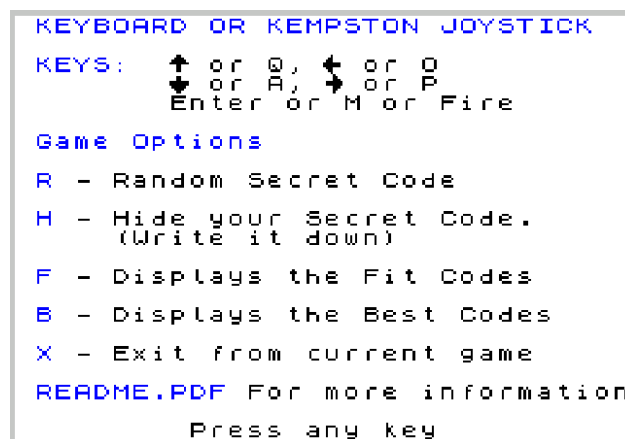


Instructions. Use of the keyboard and Kempston joystick

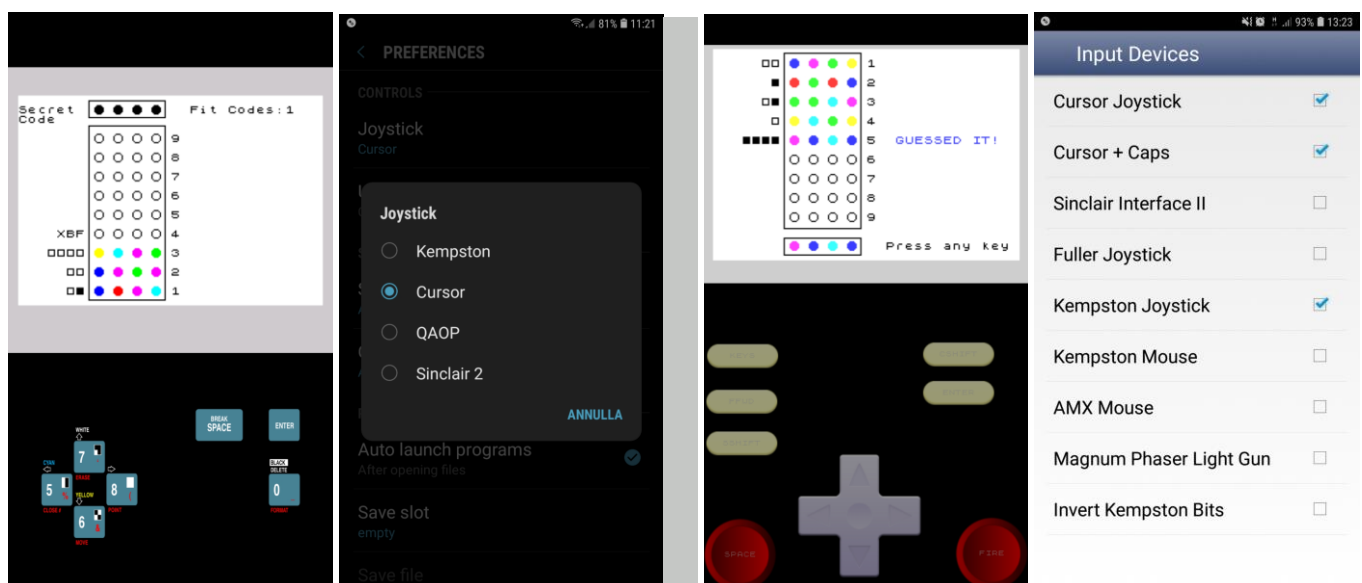
To select the colored pegs and program options, use the cursor keys or the Kempston Joystick.

The choice has to be confirmed with "Enter" or the Joystick "Fire" button.

Playing with a Spectrum with rubber keys it is better to use the alternative keys: Q A O P as cursor keys and M as ENTER key



Using an emulator of the ZX Spectrum for Android or IOS it is preferable to choose the "cursor keys" as input device in the preferences. The following image refers to: "Unreal Speccy Portable" and "Speccy" for Android.



Load and run the program.

For ZX Spectrum emulators, open the auto-starting file MMEMUXX.TAP (XX is the version).

It has been tested with Fuse for Windows, Speccy and Unreal Speccy for Android.

For the real ZX Spectrum, use the command LOAD "" and then play the MMVAWXX.WAV file from an MP3 player / Smartphone / PC connected via Jack cable to the "EAR" port of the Spectrum. Tested on my ZX Spectrum +.

Options "F" and "B"

These options are not needed to play the game, however they can be used if you are in difficulty to find an attempt code or to improve your game strategy.

Option "F" (Fit Codes)

This option displays the possible secret codes called Fit Codes by me in the program. Among these is the real secret code.

At the top right is displayed the total of the Fit Codes and how many have been displayed.

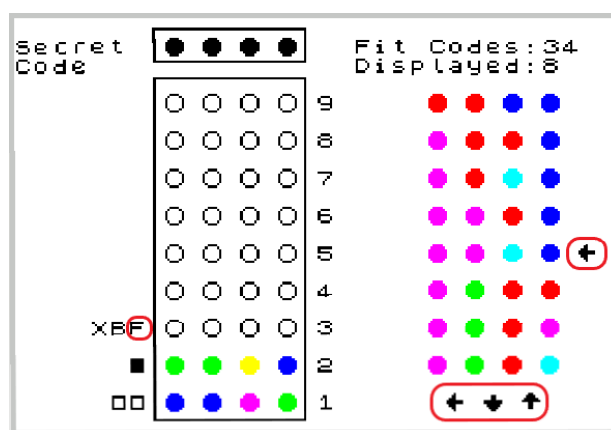
They are displayed in groups of eight. To go to the next group using the "Down" arrow key.

The "Up" arrow key enables a cursor that can scroll through the list to choose the attempt code.

The choice must be confirmed with the ENTER key.

To go back, press left arrow key.

These options are not needed to play the game, however they can be used if you are in difficulty to find an attempt code or to improve your game strategy.



Option "B" (Best Codes)

This option displays the Best Codes that are identifies by an algorithm of the program.

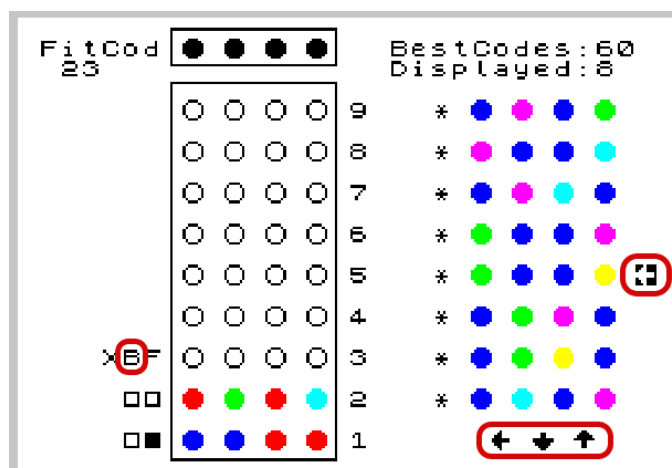
While the algorithm processes the codes, a counter indicates the progress.

Playing the Best Codes, in probabilistic terms, reduces the number of attempts to guess the secret code.

If the Best Codes are marked with an asterisk, they are inconsistent, that is, they cannot be the secret codes but still offer better chances of reaching the secret code with fewer attempts than consistent codes.

At the top right there is the total of the Best Codes and how many have been displayed.

To navigate between Best Codes and make your selection follow same instructions as per the "Fit Codes".



How the "Best Codes" algorithm works

We start to understand how Fit Codes are identified with the program that plays as codebreaker:

- 1) At the beginning the program generates all the 1296 codes and it keeps in memory. At this stage these codes are all Fit Codes.
- 2) At the first attempt, the program selects a Fit Code randomly except for codes with a color repeated 3 or 4 times. (They are the worst codes to start).
- 3) After the codemaker assign the Key Code, it compares its attempt Code with each one of the Fit Codes. All those that generate a different key code than the one received, are discarded.
- 4) The program plays a Fit Code.
- 5) The program repeats the steps from point 3).

Now, given that each Fit Code has the same probability as the other Fit Codes to guess the Secret Code, if you do not guess it, according to the Key Code received, many or less Fit Codes will remain for the next attempt.

The goal of the algorithm is to reduce as much as possible the number of Fit Codes in probabilistic terms.

Obviously, the luck component cannot be eliminated and maintains a significant role.

First version of the algorithm:

I developed this version in 1985 and I called it: "Simula". I have included the old listings in the paragraph: "ZX Master Mind: The short story about how it went". In the current program, the algorithm has been improved.

Here's how it works.

A copy of the Fit Codes list is used as a list of possible Secret Codes.

For each Fit Code a score is associated which is determined as follows: the program simulates to play the first Fit Code and the Secret Code is assumed to be the first on its list. Based on the generated Key Code, all Fit Codes compatible with it are counted. The number obtained is added to the score of the first Fit Code.

This operation is repeated for the second secret code of the list obtaining a new number to add to the first Fit Code score.

Once all the Secret Codes are finished, program moves to the second Fit Code and repeat the simulation for all the Secret Codes to obtain the score also for the second Fit Code.

The above procedure is repeated for the remaining Fit Codes.

Upon completion, the Fit Codes with the lowest score will be elected Best Codes.

If there are multiple Fit Codes with the same minimum score, they will all be elected as Best Codes.

This algorithm guesses the Secret Code on average of 4.54 attempts. Result from 1000 games played.

Improved version:

A copy of the list of Fit Codes is made and this is considered the list of possible Secret Codes.

Now it is simulated to play the first Fit Code of the list and it is assumed that the Secret Code is the first on its list. In this case, since the two codes are identical, the Key Code will be 4 black.

For each Fit Code, a table like the one in the figure below is defined in advance. For each combination of key code there is a box that is set to zero.

In this case, the "4 black" box will be increased by one.

	0 Black	1 Black	2 black	3 Black	4 Black
0 White	0	0	0	0	0
1 White	0	0	0		
2 White	0	0	0		
3 White	0	0			
4 White	0				

The operation is repeated keeping the first Fit Code fixed and moving on to the second Secret Code.

Here too, based on the Key Code obtained, the corresponding box will be increased.

The process is repeated for all Secret Codes.

At the end, the values in the boxes will indicate the number of Fit Codes that would remain if the first Fit Code was played and the Key Code was the one of the box itself.

For example, if the box with coordinates: 2 Blacks and 2 Whites contains 3, it means that by playing the first Fit Code and receiving 2 Blacks and 2 Whites as key code, the remaining Fit Codes for the next attempt will be 3. (Obviously if the secret code is not guessed)

At this point, the algorithm associates the maximum value (MAX) found in the table to the first Fit Code.

MAX is therefore the maximum number of Fit Codes that can remain if the first Fit Code is played.

The algorithm now repeats the same process for the next Fit Code, obtaining its MAX and for all other Fit Codes. Now all Fit Codes will be checked. The one with the lowest MAX value will be elected Best Code.

If other Fit Codes will have the same value of MAX, these will also be Best Codes.

Instead of fit codes, the algorithm can consider all 1296 codes by putting the Fit Codes in competition with the inconsistent codes, (codes that cannot be the secret code). Well, sometimes the inconsistent codes have a lower MAX than the Fit Codes and therefore they can be played by leaving a smaller number of Fit Codes for the next attempt.

If part of the Fit Codes and some inconsistent codes will receive the same MAX value, only the first ones will be elected as Best Codes because they also have the possibility to guess the secret code.

Since the processing time increases based on the number of codes processed, it is necessary to limit the use of the algorithm to avoid excessive waiting times by boring the player.

If the program is a decoder, the algorithm is executed if the fit codes are less than or equal to 317.

If the fit codes are less than or equal to 50, the algorithm considers all 1296 codes for processing.

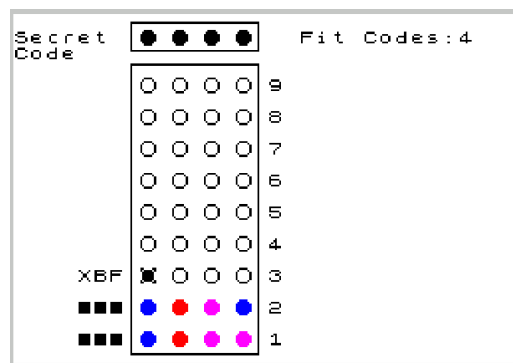
Same mode as above when the player is the decoder and select option "B".

By choosing as the first attempt, a code that is not a color repeated 4 times, can remain a maximum of 317 Fit Codes and therefore, unless the case above, the algorithm can be executed starting from the second attempt.

When, in subsequent attempts, the number of Fit Codes drops, 50 will be a good compromise to extend the use of the algorithm to all 1296 codes with the possibility of having inconsistent codes as Best Codes.

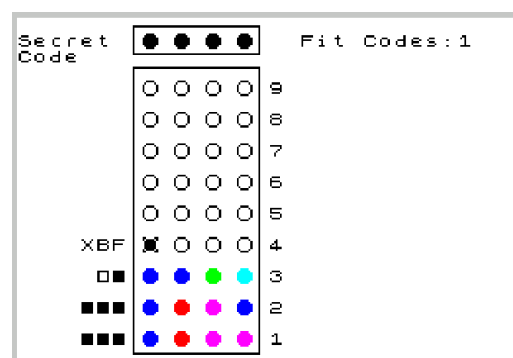
Example of using inconsistent code.

Assume that you have come to this game situation:



It doesn't look bad, 3 blacks on the first attempt and also on the second that confirm that the first 3 colors are the ones that received the black pegs. At this point just repeat the code changing the last color with one of the 4 possible. By doing this however, you have a 25% chance of guessing the secret code on the third attempt, but if you are unlucky, you will guess it on the sixth attempt.

If instead you play an inconsistent code as in the figure below:

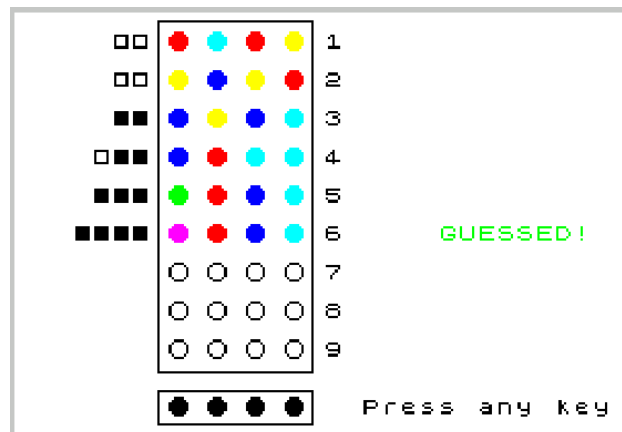


You will have the certainty of guessing the secret code on the fourth attempt.

On the fifth attempt, if in the same situation the first code played is composed by 4 different colors.

How many attempts does the program need to guess the secret code?

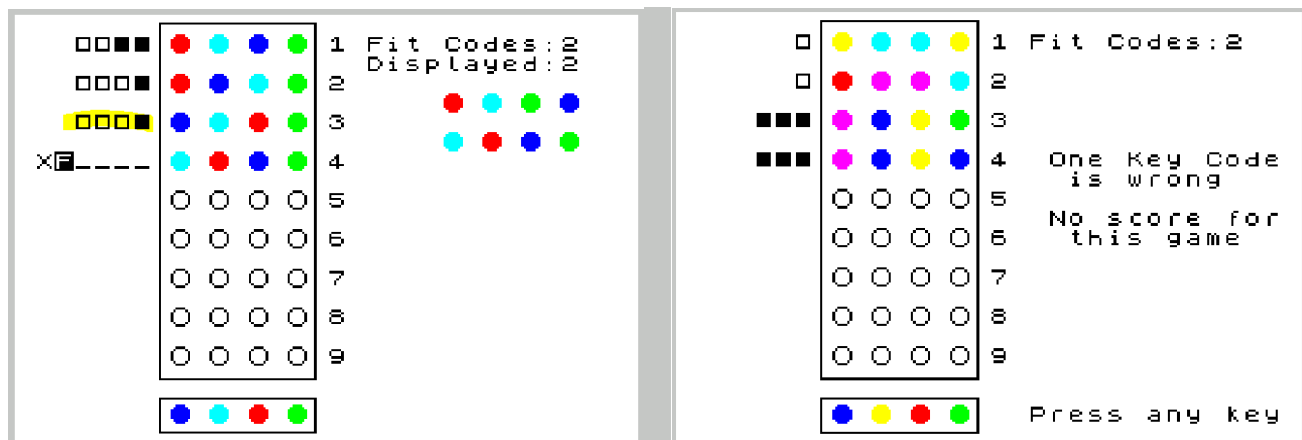
At most 5. The average is 4.47. The program could do better but the waiting times for code processing would be longer. In rare cases and only if the initial attempt is of type 1123, (single repeated color), 6 attempts may be necessary as in the example below.



Initial attempts of type 1112 and 1111 are those that involve a greater risk of guessing the secret code on the sixth attempt or more and therefore are never played by the program.

Can you cheat by giving wrong key codes?

In the image below, the program guesses the secret code on the third attempt but receives three whites and one black in response. Since this key code still leaves two possibilities, the program proceeds with the next attempt. In the case instead, an incorrect key code leaves no possible attempts, the game stops and a message will warn that you gave at least one Key Code wrong.



"ZX Master Mind"; The short story about how it went

In the 80s, microcomputers such as ZX Spectrum and Commodore 64, gave my generation the possibility of having a computer at home without spending large amounts of money and being able to learn coding, even if the great diffusion of this type of computer was more related to the possibility of used as a game console. Games were sold in the original version or recorded on audio tapes, attached to magazines that were purchasable at newsstands. I started writing the Master Mind program in 1985 when I was a student in Computer Science high school. I found it challenging to implement the logic of the games in the programs such as Connect 4, Battleship and Tetris. I wrote the part of the logic of the games, the one that gave me the most satisfaction, but then I used to lose interest and the games remained incomplete.

Master mind was the one that took me the longest time because the goal was to create a program that could be a better player than a great human player. I was excited.

I could have tried to learn games strategies and I could become better than my friends, if not the best. :-)

I didn't know how far I was going to go, but I soon realized that pursue my goal would not be easy.

The developed Basic code was too slow for my elaborations, so I decided to convert some parts in Assembler.

Another challenge was memory management, I had to be able to keep in memory the program in basic, the source in assembler and the compiled, two assemblers: Zeus, which I used to write the assembler and Champ, for the phase of debugging instructions per instruction and finally the 1296 combinations which occupied about 13K.

At the end, I got the "heart" of the program, consisting of Assembler routine, but such as other programs it remained incomplete.

From that moment, I didn't do anything again and the only time I saw Spectrum was when I had to move it from basements to attics. "Sooner or later I would like to switch it on and do something," I thought.

Finally, the right time came: while I was arranging old paperwork, I found old listing of programs including those one of Master Mind assembler routines.

I hoped to start again from the program that I left but turning on my Spectrum, I immediately realized that it had some problems, so I had to give up.

So, I decided to find a Spectrum emulator for PC, in order to rewrite the program starting from the routine sheets that I had found.

Here I show you the routines. There is also data codes structure and how I had organized the memory to have source code, the compiled and the two assemblers at the same time:

① CP Inizializzazione del campo "PUNTATORE" ⑤

ORG 2
START EQU 4000 CP EQU ...
ADDCP EQU START+10

LD BC, 1287
LD DE, 10 ; DE = incremento
LD IX, CP-12 ; IX ← al campo puntatore
LD HL, CP-10 ; HL = valore base da mettere nel campo puntatore

Loop ADD IX, DE
ADD HL, DE
LD (IX), HL
LD (IX+1), HL
DEC BC
LD A, B
OR C
JR NZ, Loop
LD (IX), FF
RET

PUNTATORE

① CP Inizializzazione campo "POSIZIONE COLORI" ③

START EQU 4000 ORG 3 40070 = 3C72-189
ADDCP EQU START CP EQU 43000 = A7F8

ORG 3
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 33
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 35
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 37
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 39
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 41
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 43
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 45
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 47
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 49
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 51
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 53
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 55
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 57
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 59
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 61
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 63
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 65
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 67
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 69
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 71
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 73
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 75
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 77
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 79
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 81
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 83
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 85
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 87
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 89
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 91
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 93
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 95
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 97
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 99
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 101
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 103
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 105
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 107
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 109
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 111
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 113
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 115
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 117
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 119
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 121
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 123
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 125
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 127
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 129
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 131
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 133
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 135
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 137
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 139
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 141
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 143
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 145
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 147
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 149
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 151
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 153
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 155
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 157
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 159
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 161
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 163
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 165
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 167
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 169
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 171
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 173
LD D, 20
LD E, 20
LD B, 20
LD C, 40

ORG 175

SIMULA

Calcolo il peso di ogni CP

ENTRATE USCITE MODIFICA tutti i registri e comp. PESO

```

ORG9 EQU
LD DE,FOOF CP EQU
OXX EQU
ORG6 EQU
PUSH IX ORG7 EQU
LD IX,CP
NEXT3 LD IX,(CP-2)
Next2 CALL ORG6
LD AL,C LD A,C ;L=N^2X
LD (X),A CALL ORG7
LD H,A LD A,C ;H=N^2OX
LD (OX),A
PUSH IX
LD IX,(CP-2)
CALL ORG6
LD A,(X)
LA A,L
CPH
UR N2,Negative
CP 4
UR Z,Negative
CALL ORG7
LD A,(OX)
CP C
UR N2,Negative
LD B,(IX+6)
LD C,(IX+7)
INC BC
LD (IX+6),B
LD (IX+7),C
Negative LD A,FF
XOR (IX+8)
UR Z,Exit1

```

Flowchart:

```

graph TD
    DESI --> FOR1[FOR IX=1 TO CP]
    FOR1 --> TEST1[TEST 1/2]
    TEST1 --> X4C1["(X)+C1  
(OX)+C2"]
    X4C1 --> SALVAIX[Salva IX]
    SALVAIX --> FOR2[FOR IX=1 TO CPR]
    FOR2 --> TEST2[TEST 1/2]
    TEST2 --> PESO["PESO(IX)=PESO(IX)+1"]
    PESO --> NEXTIX[NEXT IX]
    NEXTIX --> RIPRISTIAIX[Ripristia IX]
    RIPRISTIAIX --> NEXTIX
    NEXTIX --> NEXTIX
    NEXTIX --> NEXTIX
    NEXTIX --> RET

```

CONTINUA

SIMULA

(continuazione)

```

LD B,(IX+8)
LD C,(IX+8)
PUSH BC
POP IX
UR Next1
Exit1 POP IX
LD A,FF
XOR (IX+8)
UR Z,Exit2
LD B,(IX+9)
LD C,(IX+8)
PUSH BC
POP IX
UR Next2
Exit2 LD A,FF
XOR (IX+8)
UR Z,FINE
LD BC,10
ADD IX,BC
UR Next3
POP IX
FINE RET

```

Flowchart:

```

graph TD
    Exit1 --> LD_A_FF[LD A,FF]
    LD_A_FF --> XOR_IX8[XOR (IX+8)]
    XOR_IX8 --> UR_Z_Exit2[UR Z,Exit2]
    UR_Z_Exit2 --> LD_B_IX9[LD B,(IX+9)]
    LD_B_IX9 --> LD_C_IX8[LD C,(IX+8)]
    LD_C_IX8 --> PUSH_BC[PUSH BC]
    PUSH_BC --> POP_IX[POP IX]
    POP_IX --> UR_Next2[UR Next2]
    UR_Next2 --> Exit2
    Exit2 --> LD_A_FF2[LD A,FF]
    LD_A_FF2 --> XOR_IX82[XOR (IX+8)]
    XOR_IX82 --> UR_Z_FINE[UR Z,FINE]
    UR_Z_FINE --> LD_BC_10[LD BC,10]
    LD_BC_10 --> ADD_IX_BC[ADD IX,BC]
    ADD_IX_BC --> UR_Next3[UR Next3]
    UR_Next3 --> POP_IX2[POP IX]
    POP_IX2 --> FINE_RET[FINE RET]

```

CP < CPR

PESO

MASTER MIND

STRUTTURA DEL FILE

START=4000 ABEO

START	I NIBBLE	II NIBBLE	III NIBBLE	IV NIBBLE	POSIZIONE COLORI	NUMERO COLORI
START+1	II NIBBLE	IV NIBBLE	1 VI			
START+2	II NIBBLE	IV NIBBLE	2 E2			
START+3	II NIBBLE	IV NIBBLE	3 E3			
START+4	II NIBBLE	IV NIBBLE	4 E4			
START+5	II NIBBLE	IV NIBBLE	5 E5			
START+6	BYTE	6 E6	X			
START+7	BYTE	7 E7	OXX			
START+8	ADDRESS	8 E8				
START+9	ADDRESS	9 E9				
START+10	OS NIBBLE	IS NIBBLE	10 E10			
START+11	11E NIBBLE	11E NIBBLE	11 E11			
START+12	2E NIBBLE	2E NIBBLE	12 E12			
START+13	3E NIBBLE	3E NIBBLE	13 E13			
START+14	4E NIBBLE	4E NIBBLE	14 E14			
START+15	5E NIBBLE	5E NIBBLE	15 E15			
START+16	WORD	16 F0				
START+17	WORD	17 F1				
START+18	ADDRESS	18 F2				
START+19	ADDRESS	19 F3				

VARIABILI

RECORD

PUNTORE

0-E0 1-E1 2-E2 3-E3 4-E4 5-E5 6-E6 7-E7 8-E8 9-E9

10-EA 11-EB 12-EC 13-ED 14-EE 15-EF 16-F0 17-F1 18-F2 19-F3

CP I II III IV V VI 1 2 3 4 5 6 K EQU 44000

8 R M V A G B A M V A G ADDC EQU X+6 OXX EQU X+7 PUNT EQU X+8 ADDCP EQU X+10

MASTER MIND

SORGENTE

OGGETTO

FILE

37000-3088 ORG2 CP Prog Punt CP Bytes=114 ORG4 ORG5

37200-3150 PUNTORE ADDCP Bytes=184

37400 POSIZIONE COLORI ADDCP Bytes=246

37400 N°COLORI LOW ADDCP Bytes=265

37600 N°COLORI HIGH ADDCP Bytes=301

38000 TEST1 N°X IN: IX+CP IX+C OUT: C+X

38200 TEST2 N°OX IN: IX+CP IX+C OUT: C+X

38200 CPR IN: (Q), (X) (OX) ADDCP X, OXX ORG6 ORG7 Bytes=533

40300 SIMULA ADDCP X, OXX ORG6 ORG7 Bytes=526

40850 TROVA C ADDCP C Bytes=272

41250 CLS PESI ADDCP C Bytes=138

43000-47F8 ORG1 Bytes=13

43012-4804 ORG2 Bytes=32

43051-4323 ORG3 Bytes=183

43258-48FA ORG4 Bytes=45

43314-4932 ORG5 Bytes=55

43324-496E ORG6 Bytes=35

43401-4999 ORG7 Bytes=113

43532-4A0C ORG8 Bytes=24

43653-4A85 ORG9 Bytes=112

43781-4B0F ORG10 Bytes=53

43910-4B86 ORG11 Bytes=23

43000-4385 START C

43006 START+1 X

43007 START+2 OXX

43008 START+8 PUNT

43010 START+10 CP1

43012 START+18 PUNT

43014 CP2

43016 PUNT2

43030 CP3

43038 PUNT3

43040 CP1295

43042 PUNT1295

43044 CP1296

43046 FFXX

MEMORY MAPPING

23000 CHAMP

36582

37000 MASTER SOURCE

42800

43000 MASTER CODE

43990

44000 MASTER FILE

44000 MASTER

57344-5000 ZEUS

65278

I was really out of practice and it took me a long time to regain confidence with the Assembler and to understand the logic of the routines I had written.

Recalling the reasoning made 35 years earlier was as exciting as reviewing photos of those times. Then I tried to recreate the environment I had on the Spectrum but this time with the FUSE emulator and Zeus Assembler + CHAMP in the TZX version; Zeus for programming and CHAMP for debugging.

After a while I realized that this procedure was too laborious and slow and so I searched, and fortunately found, an IDE (Integrated Development Environment) to be used on a Windows PC called "BASIN".

The rest is recent history of many nights spent writing and above all rewriting, improving, speeding up the code and making it fit in the 48K.

Many thanks to the authors of BasIn. Without this tool, I don't think I would have been able to finish this project.

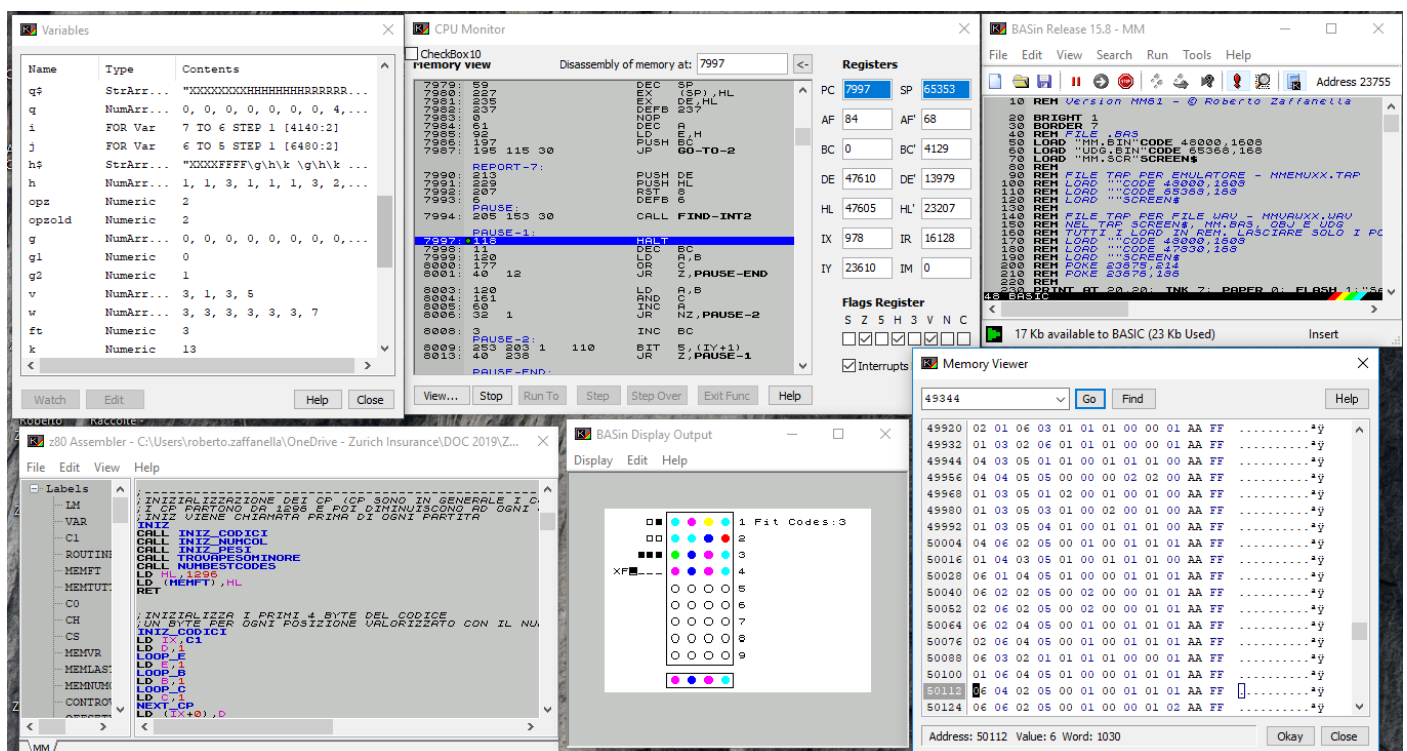
I would highly recommend it to anyone who wants to develop both in Basic and Assembler.

In case, however, I recommend version 15.8. With later versions I have encountered problems.

Here the download link:
<https://sites.google.com/site/ulaplus/home/zx-spin-and-basin> 15.8

Unzip in a folder and it is ready; does not require installation.

Below is a screenshot shows you the program running. You can see the lists in Basic and Assembler, the program running, the value of the variables, the contents of the memory and the execution of the object code:



Questions, bug reports or suggestions are welcome. Write to me at zaffaroby@gmail.com

Have fun!! ☺