A Solution of the Mastermind Board Game in Scratch Suitable for Education - Results of the Preliminary Case Study

TOMAS HORNIK, PETR COUFAL,
MICHAL MUSILEK AND STEPAN HUBALOVSKY
Department of Cybernetics
University of Hradec Kralove
Hradecká 1227, Hradec Králové
CZECH REPUBLIC
tomas.hornik@uhk.cz, petr.coufal@uhk.cz,
michal.musilek@uhk.cz, stepan.hubalovsky@uhk.cz
https://www.uhk.cz/en-GB/UHK/

Abstract: - The article is a case study of a specific problem - popular board game Mastermind and its solution in Scratch, visual online programming language. A preliminary version of this paper was presented at APSAC 2017. Emphasis is put on the educational perspective both in the logic behind the solution itself and on the way the problem can be presented to elementary school pupils. The article is focused on logical explanation of the solution and on work with several specific programming elements, like IF-ELSE conditions, data structures and simple bug hunting feature. The difficulty is suitable for elementary school pupils as a complex task meant for superior individuals or a group of pupils. It was successfully tested as a small scale preliminary study conducted on pupils aged between 12 and 15 at an extracurricular group. The results of the qualitative research are presented at the end of this paper.

Key-Words: - Algorithmic Thinking, Algorithm Development, Education, Elementary School, Mastermind Game Solution, Programming, Scratch, Educational Programming Languages

1 Introduction

The teaching of algorithm development and programming is no longer a domain of highly specialized and technically oriented high schools and universities, but progressively appears at elementary schools as well. There were some attempts on children programming languages as far back as in 1960s, for example KAREL or LOGO [1]. Both of them are still used, albeit in a limited extent. With the sharp increase in the amount of technology involved in every day life of ordinary people, the need for teaching of understanding how the programs work is also heightened and text based programming languages are being at least accompanied (if not replaced) by visual languages, such as Blockly, Scratch, Snap! (formerly BYOB), KODU, LEGO Mindstorms NXT-G, and others.

All these languages are very robust and can be used for creation of quite complex codes. Following chapters deal with a solution of board game Mastermind in Scratch. The solution goes beyond merely re-creating the game, because it also includes an algorithm by means of which the computer can solve every game of the original Mastermind (six colors, four positions) in maximally ten rounds. Selected solution is fully explained in chapter three and specific problems are pointed out from the educational point of view in chapter four.

2 Problem Formulation

Mastermind is a commercial logical board game for two players that develops logical thinking. One player hides a secret code (combination of colors on certain positions), while the other one is trying to find it out. Every guess is evaluated by black and white pegs indicating how close to the hidden code it was. Black peg means that the guessing player guessed a color correctly and even put it in the correct place. White peg means that a guessed color is correct, but it is misplaced. Everything is made more difficult by the fact, that order of black and white pegs evaluating the guess is random and has no connection with the actual position of colors within the guess (e.g. a white peg on the first place in the evaluation does not mean, or does not have to mean, that the color on the first place is correct, but should be elsewhere).

The guessing player loses when he or she reaches the end of the playfield by exceeding given amount

of possible guesses. There are variations of the game with more colors or positions, and in some versions it is allowed to use one color on multiple positions, or to use no colored peg at all, thus omitting a spot within the hidden code. Both the possibility of multiple occurences of a single color and ommission of a spot in the secret code are decisions made by the players themselves, because these changes make the game increasingly difficult and at the same time do not require any physical alterations in the implementation of the game itself. The detailed explanation of the rules including examples exceeds the scope of the paper and can be found for example on official website of Pressman, one of the manufacturers producing the game [2].

Even though the game was invented in early 1970s, it is still an attractive topic for mathematicians. Every modification of the game usually requires different approach from the guessing player and from the scientific point of view the game was even proved to be NP Complete problem. [3][4] New solutions trying to minimize the number of necessary rounds are still being introduced. Donald Knuth proved it is possible to solve the game in five or less rounds. [4] Temporel and Kovacs created a heuristic hill climbing algorithm, that induces new potential guesses with minimum computation. [5] They stated, that there is one more aspect, beside the minimal number of rounds necessary to solve the code, that should be minimised. This aspect is the amount of combinations evaluated before a new guess is chosen. They assess the fitness of every evaluated combination and each new guess, based on a relatively simple mathematical formula, must be consistent with all previous guesses. [5]

It is possible to create even so complex solutions in Scratch, mainly because of its overall robustness, which opens new possibilites for further research. However, it is not advisable to implement so difficult ideas as examples in real world teaching practice. All these solutions are not suitable for educational purposes and neither are the more difficult variants of the game. This paper deals with the original version consisting of six colors and four positions, where multiple use of the same color is allowed but empty spots are not. Solution presented in following chapter is slow regarding the average and maximal number of rounds, but it is very straightforward from the point of view of logic and requires minimal computational power.

3 Problem Solution

All the aforementioned solutions require deeper understanding of mathematics and an ability of advanced abstract thinking. Following solution is intended for pupils at elementary schools in the age of 12 to 15. Pupils at this age are already able to think in abstract terms [6], however this ability is not yet fully developed. The program itself is rather complex for an elementary pupil, but it is relatively easy to explain.

The solution was implemented in Scratch 2.0 online. When working with Scratch, it is necessary to keep in mind certain aspects of the language, as for variables. There are just general variable and a simple list. Another trait connected with Scratch is its realization in Flash. Even though Flash is still a rather robust tool on computers, utter absence of native support on Android and iOS phones and tables makes it "obsolete" for this particular group of users. Although it may seem superficial, the inability to use pupils own mobile phones and tablets is something, that should be taken into consideration when speaking of the education process. [7] This deficiency is only temporary and the problem will be addressed by Scratch 3.0 based on HTML-5. [8]

3.1 Algorithm Division

Compartmentalization of a given problem is one of the first steps necessary for the algorithm development in any programming language. This skill is also easily transferable into every day life, since everyone is dealing with more or less complex problems on a daily basis. By learning the skill of compartmentalization pupils are acquiring an ability to subdivide intricate tasks into more manageable fragments. This should also be the primary goal in teaching of programming at elementary schools. There is no need to turn every pupil who encounters programming at elementary school into a professional programmer, just as it nonsense to presume that everyone who learns basics of Physics can and will be a theoretical or nuclear physicist.

The algorithm itself can be very roughly divided into following steps/categories:

1. Decision of how to store the data (selection and naming of basic variables, creation and naming of necessary lists)
2. Preparation of a new game (erasing all the lists, setting all the variables on their default value, random generation of a new hidden code and initial guess)

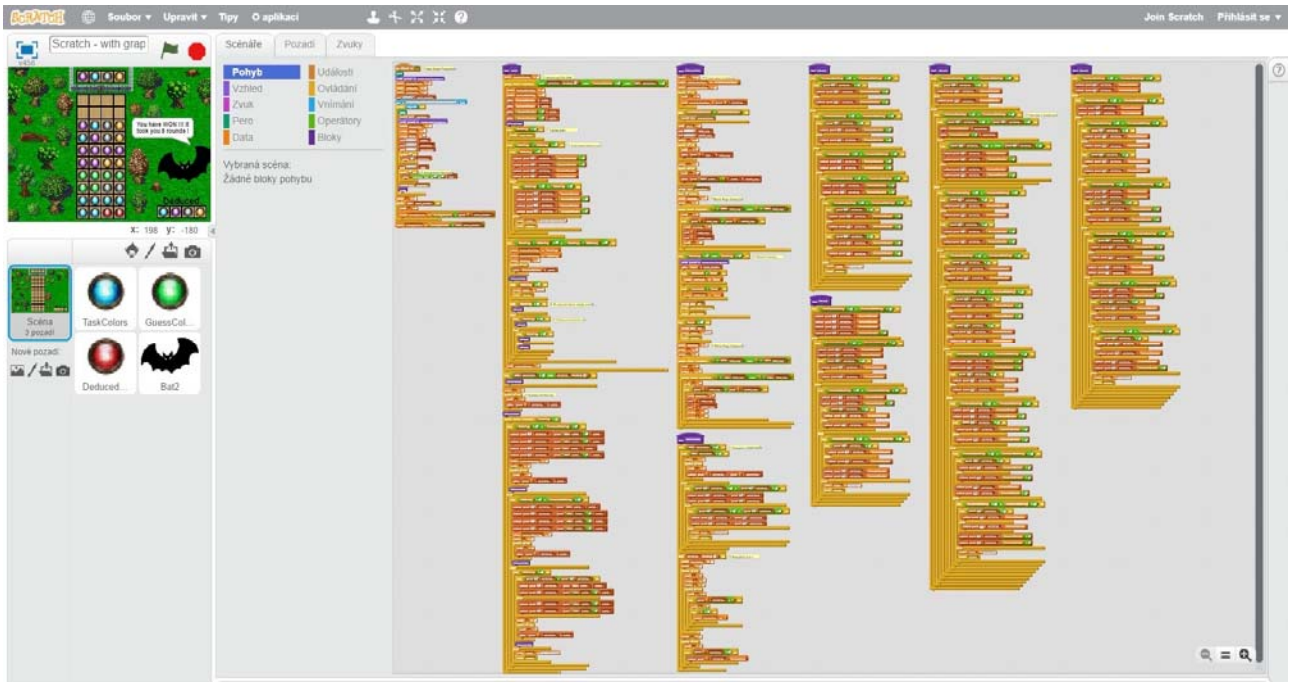

Fig. 1. Overall length of the final program created in Scratch.

3. Creation of universal evaluation custom block (a process called after every guess that tells the number of black and white pegs according to the guess and decides victory or defeat)
4. Enumeration of all the possible decisions (based on two rows of guesses and their evaluation by black and white pegs)
5. Addressing of exceptions (one specific combination of black and white pegs and the possibility of having 3 or even 4 same colors)
6. Addition of graphics (background and colored dots representing guesses)
7. Final Calculations (counting number of games and average rounds within them)

3.2 Data Structures and Conditions

The program contains seven lists and sixteen variables - for standard game calculations, round calculations, exception handling, error notification, graphical representation and auxiliaries utilized in computations throughout the whole code. From the point of view of pupils, this is a lot of different and confusing numbers, but every variable and list has specific purpose. In spite of full implementation of global and local variable system in Scratch, it was avoided and all the variables are global.

The flow of the program is controlled by four basic structures - loops with counter, repeat-until loops, IF conditions and IF-ELSE conditions. The difference between IF and IF-ELSE conditions is relatively simple, yet the pupils have hard time deciding which one of these they should use under given circumstances.

The decisions in guessing colors and their order use following logic - the solver inputs two dots of one color and two dots of different color. Six colors are represented by numbers from 1 to 6, so the first line is 1122. If this line has at least one hit (either white or black), the solver inputs four dots with only color 1 and from there he or she decides what is in the hidden task. If the first row from the guess has zero white and zero black pegs, the two colors are eliminated and second line made of just one of the two colors is skipped. The process is repeated three times. Despite having some exceptions, the logic behind the solution is simple enough to be fully understood by the pupils after the first explanation. All the possible combinations are shown bellow.

Table 1. Possible combinations of a two line guess. Second line made of only one color can be evaluated with black pegs only and the decision making starts from there. Black pegs are B, white pegs are W and the number of them is written before the letter (two black pegs = 2B).

2nd line	Possible combinations with the first line								
0B	1B	0B	2B	0B	1B				
	0W	1W	0W	2W	1W				
1B	1B	1B	2B	0B	0B	0B	2B	1B	3B
	0W	1W	0W	1W	2W	3W	1W	2W	0W
2B	2B	0B	1B	3B	0B	1B	2B		
	0W	2W	1W	0W	3W	2W	1W		
3B	3B	2B	1B	1B					
	0W	0W	2W	1W					

Fig. 2. Side by side comparison of the game. Graphical visualization is on the right side, whereas hidden graphics with most of the variables and lists shown is on the left side.

Nesting of IF-ELSE blocks is in this case far superior to basic IF conditions not only because when the correct option is found the unnecessary code section is skipped, but also because if the correct option is not found, the ending section can terminate the program and write an error message. This error message can be changed in context of the place where it happened within the program and thus it can make bug hunting process (= tracing errors within the program [9]) much easier.

3.3 Weaknesses of the Algorithm

In contrast with solutions of other authors (see chapter 2) this algorithm lacks the robustness and takes a lot of guesses. Where other complex solutions get close to four rounds per average game and maximum number of rounds is five (see chapter 2), this solution takes seven rounds in average and in the most unfavorable case it is ten rounds. Nevertheless, the algorithm still manages to beat the game every time.

Another weakness lies in its inability to adapt to more difficult Mastermind versions. Addition of more colors, more positions or reduction of maximum rounds before defeat causes the algorithm to fail and the algorithm does not work.

3.4 Graphical Representation of the Game

Implementation of graphical elements into the program is relatively simple if the pupils understand the program they wrote. This teaches them that readability and logical layout of the program really is important. It also shows how boring computation can be relatively easily turned into a full scale game given just a few simple sprites (2D pictures representing objects from the game). When they see with their own eyes the difference (see figure 2) and how difficult it is to create the code part of the program, they can realize that games are equal part of graphics and coding.

4 Stressed Educative Elements of the Solution

The aforementioned solution is a project suitable as the final program presented at the end of a course. While creating a project of such a complexity pupils have to demonstrate deeper understanding of all the programming principles they had been learning throughout the given course as well as the ability to merge them within a single program. The teacher can give the pupils advice during any of the stages; however, it is advisable to show how should the solution work on an exemplary game. This way all the pupils can work towards the same goal and can help one another. The first step in the development of the algorithm was for pupils to segment the whole task into several more or less independent sub-tasks. This skill is the most important one for pupils' every day life. Teaching of programming presents an opportunity to show students basic principles of compartmentalization and its impact on efficiency. It also teaches sequential thinking - finding the beginning and moving from there in premediated manner.

Although the program can be created as a single procedure, the resulting outcome is not legible and it is apt to take advantage of the possibility to create Custom Blocks. The Custom Blocks are almost

necessary for the Evaluation procedure, but they are also beneficial for making the program well arranged. For the same purpose it is also good to insist on employment of comments.

The solution of the game needs 7,05 rounds in average to win the game (based on a cycle of 10 000 consecutive games), but the logic behind it is very easy to explain and presents interesting opportunities for teaching of algorithm development. Even though the amount of possible decisions needed to finish the game is very limited (see table 1) pupils must enumerate all of them, otherwise the program does not work (or at least not always). This forces the pupils to think from different points of view. The pupils have a strong tendency to create a solution functional for the problem only within current circumstances. In other words - if it works now, it will work every time. The necessity to consider all the possible circumstances (like different values stored in variables) is not taken into account and the given section of code is perceived as unwaveringly correct.

Same problem also creates an opportunity to show the pupils how to implement basic bug-hunting tool. After the pupils find out on their own how difficult it is to search for bugs in the whole program, they appreciate something, that would tell them exactly in which section of the program the problem occurs (see chapter 3.2).

When a pupil reaches fully functional algorithm, he or she can add an option to "re-play" the game as many times in a row as the user wants. This allows to test the solution exhaustively by simply letting it repeat itself several hundred or even thousand times. Such a loop requires enormous amount of time in Scratch's standard mode, but there is a Turbo Mode, which "runs the project extremely fast, having minimal to no wait between blocks." [10]

Faster pupils can also implement round counter and calculation for maximal and minimal length of a round occurring during the long reiterative run of the program. Based on these numbers they can also add calculation for average round length. Since the Mastermind Game is a complex problem (as stated in chapter 2), the pupils are welcome to try and create their own solution, after they successfully finish this one. From this point of view, the creation of a new program is more of a brain teaser than purely algorithmic problem and as such, possibly only the gifted pupils can do so.

Scratch also proved to be robust enough for implementation of far more complex solutions for the Mastermind game created for example by Knuth or Temporel and Kovacs. These solutions can be shown to pupils in order to prove that even though different solutions may lead in the end to the same results, but the path itself can be entirely different in approach, complexity, demands put on the computer and given programming language as well as in the overall efficiency of the final program.

5 Preliminary Qualitative Case Study

Further testing was conducted as a small scale preliminary study with pupils aged between 12 and 15 at an extracurricular group. Extracurricular groups are voluntary clubs organized by a school or some (usually) non-profit organization owned by a city or a district that offers an option to further engage in different areas of interest. With respect to their inherent nature, resulting groups are usually non-homogenous and only link between individual participants can be the subject matter.

The testing group was composed of seven pupils, two girls (age 12 and 14), and five boys (age 12, 12, 14, 14, 15). Since the group is organized by a school (Elementary School Uprkova 1, Hradec Kralove, Czech Republic), all the pupils were from the same school. The pupils ranged from 6th grade up to 8th grade and they differed in their level of abilities, from slightly below average up to very gifted one. Distinct differences could also be seen in their attention span, persistence as well as in the overall way of thinking (way they approached same problem entirely differently).

The trial showed that understanding of the logic behind the solution was without any problem. All the pupils were able to grasp how the presented solution works. Determination of steps necessary for the solution of the problem was skipped by six of the seven pupils, who just unrestrainedly started to try putting something together without any prior consideration or planning. The seventh pupil (8^{th} grade boy, aged 15, very apt) planned the work ahead and needed minimal to no help during the whole course of the work. The abovementioned six pupils got quickly entangled in their chaotic solution and had to start over. The steps were identified collectively as following: 1. start state of the game (preparation of variables and/or restart of the game); 2. random generation of a secret code; 3. black and white pegs evaluation; 4. generation of a new guess; 5. evaluation if the game is won or lost.

Even after individual steps were specified, pupils with short attention span and/or low persistence had a very strong tendency to skip from step to step without properly completing them. This led to a massive amount of unnecessary mistakes originating from unfinished parts of the code. Letting the pupils work in pairs (except the gifted one) substantially

helped, since they corrected each other's mistakes and in most cases, they were also able to stay on a given step until its completion.

Despite overall logical simplicity of the solution, orientation in a program this vast proved to be a problem for most of the pupils. A program of such magnitude proved the pupils how important is to divide it into logical parts using Custom Blocks and how important is to comment individual sections of code. Also enumeration of all the posible options when deciding the next guess took a massive amount of time, as expected, and showed how invaluable can a simple bug hunting extension be. Nobody was able (or willing) to do the extra work suggested at the end of chapter 4 for faster pupils and as such there is no way to evaluate how usable are the ideas from the educative point of view.

Another important observation is, that pupils could not work only with variables and lists. They all had to immediately transfer the numbers into visible graphical representations. Also most of the reasoning was based on the graphical depiction.

4 Conclusion

In conclusion the idea of a Mastermind solution proved to be partially fit for education. Average pupils had considerable problems with most of the work and time demands were enormous. As such it is recommended to use the task only with very smart pupils who are willing to choose it as a final project and work on it in their free time. On the other hand, if the automatic preparation of the new guess is omitted, the process of creation of a Mastermind game (consisting of randomly generated invisible secret code, black and white pegs evaluation and guessing based on the user input) is an activity suitable for all the pupils and probably applicable in regular IT lessons as well. However, deployment in actual regular class is yet to be tested.

*Acknowledgments***.** The paper has been supported by Specific Research Project of Faculty of Science, University of Hradec Kralove, 2017 and by Specific Research Project of the Faculty of Education, University of Hradec Kralove, 2017.

References:

[1] Musílek, M.: *Kapitoly z dějin informatiky*. Gaudeamus, Hradec Kralove. ISBN 978-80-7435-129-7. (2011)

[2] *Mastermind Rules*, http://www.pressmantoy.com/game-rules/Mastermind_rules.pdf

[3] Stuckman, J. and Zhang, G.Q.: *Mastermind is NP-Complete*. (2006)

[4] Goodritch, Michael T.: On the algorithmic complexity of the Mastermind game with black-peg results. In: *Information Processing Letters* 109 675–678. (2009)

[5] Temporel, A., Kovacs, T.: A heuristic hill climbing algorithm for Mastermind. In: *UKCI'03: Proceedings of the 2003 UK Workshop on Computational Intelligence*, Bristol, United Kingdom, pp. 189-196. (2003)

[6] Kohoutek, R.: Kognitivní vývoj dětí a školní vzdělávání. In: *Pedagogická orientace*, 2008. Vol. 18 No. 3, pp. 3--22. ISSN 1805-9511. (2008)

[7] Rossing, Jonathan P., Miller, Willie M., Cecil, Amanda K. and Stamper Suzan E.: iLearning: The future of higher education? Student perceptions on learning with mobile tablets. In: *Journal of the Scholarship of Teaching and Learning*, Vol. 12, No. 2, June 2012, pp. 1--26. ISSN 1527-9316. (2012)

[8] *Scratch 3.0*. In: Scratch Wiki. [online]. Accessed 17 Jun. 2017. Available at: https://wiki.scratch.mit.edu/wiki/Scratch_3.0 (2017)

[9] Ganesh, S. G.: *Joy of Programming: Bug Hunt*. [online]. Accessed 17 Jun. 2017. Available at: http://opensourceforu.com/2011/03/joy-of-programming-bug-hunt/ (2017)

[10] *Hidden Features: Turbo Mode*. In: Scratch Wiki. [online]. Accessed 13 Jun. 2017. Available at: https://wiki.scratch.mit.edu/wiki/Hidden_Features#Turbo_Mode (2017)

[11] Milková, E., Petránek, K.: Programming courses reflecting students' aptitude testing and implementing learning style preferences research results. In: *International Journal of Mathematics and Computer in Simulation*, vol. 10, 2016, pp. 218–225. ISSN: 2074-1316. (2016)