VZ 2134/17

Výroční

# Závěrečná zpráva projektu specifického výzkumu na rok 2017 (jednoletý projekt) – zakázka č. 2134

**Název projektu: Výzkum možností implementace programování do standardního kurikula výuky informatiky na základní škole**

**Specifikace řešitelského týmu**

Odpovědný řešitel: Mgr. Petr Coufal

Studenti doktorského studia na PdF UHK: Mgr. Petr Coufal, Mgr. Tomáš Hornik

Studenti magisterského studia na PdF UHK:

Školitelé doktorandů: doc. RNDr. Štěpán Hubálovský, Ph.D., PhDr. Michla Musílek, Ph.D.

Další výzkumní pracovníci:

**Celková částka přidělené dotace: 127 709,-**

**Stručný popis postupu při řešení projektu, vč. přehledu realizovaných výdajů je obsahem výroční (průběžné) zprávy.**

## Popis řešení projektu

Cílem projektu specifického výzkumu je výzkum možností implementace programování do standartního kurikula výuky informatiky na 2. stupni základních škol. V rámci specifického výzkumu bylo provedeno výzkumné šetření na základních školách v Královehradeckém kraji, šetření bylo navíc rozšířeno i o základní školy v Pardubickém kraji. Na základě výsledků výzkumného šetření byla vytvořena sada metodických materiálů pro využití robotické stavebnice LEGO Mindstorms ve výuce programování a rozvoje algoritmického myšlení žáků. Dále byla vytvořena sada metodických materiálů pro výuku programovaní a rozvoj algoritmického myšlení v dětském programovacím jazyce Scratch. Součástí projektu specifického výzkumu byla publikace výsledků v časopisech a ve sbornících z konferencí EET 2017 v Praze a APSAC 2017 v Dubrovniku.

## Průběh řešení projektu

březen až květen - výzkumné šetření a průzkum

duben až prosinec - zpracování výsledků výzkumu a publikační činnost

červen až prosinec – návrh metodických materiálů

## Přehled realizovaných výdajů:

a) osobní náklady (mzdy, odměny, odvody na zdravotní, sociální a úrazové pojištění, tvorba sociálního fondu, dohody o provedení práce a dohody o pracovní činnosti) a jejich stručné zdůvodnění

**2 000 CZK** DPP Coufal, Hornik (nutné k účasti na konferenci)

b) stipendia a jejich stručné zdůvodnění

**7 000 CZK** Coufal Petr - stipendium za řešení projektu

**7 000 CZK** Hornik Tomáš - stipendium za řešení projektu

c) materiálové náklady (výdaje na pořízení drobného dlouhodobého hmotného majetku, nehmotného majetku-software, kancelářské potřeby, ostatní materiál) a jejich stručné zdůvodnění

**1 312 CZK** kancelářské potřeby

**52 633 CZK** robotické stavebnice a příslušenství

d) poplatky konference nebo výdaje na služby a jejich stručné zdůvodnění

**14 183,63 CZK** konferenční poplatek APSAC 2017 Dubrovnik

**7 669, 35 CZK** konferenční poplatek EET 2017 Praha

e) doplňkové (režijní) náklady nebo výdaje v souladu s příslušným řídícím aktem UHK

**440 CZK** pojištění zahraničí Coufal, Hornik

**350 CZK** bankovní poplatky

f) cestovné a jeho stručné zdůvodnění

**34 563 CZK** cestovné, letenky konference APSAC 2017 Dubrovnik

**558 CZK** cestovné konference EET 2017 Praha

Celkem náklady: **127 708,98** CZK

Celkem dotace: **127 709,00** CZK

**Splnění kontrolovatelných výsledků řešení**

Úplný seznam publikačních výstupů projektu-druh (dle metodiky VaV), název, autoři, rok vydání, zdroj publikování.

*Každý výstup musí být zaveden do OBD a doložen výpisem z něj s prokázanou návazností na projekt Specifického výzkum (s uvedením čísla zakázky).*

***Publikační výstup Jimp (časopis)** – časopis indexován v databázi WoS*

[1] Coufal, P., Hornik, T., Hubalovsky, S., Musilek, M., Simulation of the Automatic Parking Assist System as a Method of the Algorithm Development Thinking. *International Journal of Education and Information Technologies*. 2017, Vol. 11, No. 2, p. 37-43. ISSN 2074-1316.

***Publikační výstup D (sborník z konference)** – sborník bude indexován v databázi WoS*

[2] Coufal, P., Hornik, T., Hubalovsky, S., Musilek, M., The Development of KarelNXT Robot as a Simulation of xKarel Programming Language. Není zadáno v OBD, čeká se na indexaci.

[3] Hornik, T., Musilek, M., Coufal, P.,Hubalovsky, S., A Solution of the Mastermind Board Game in Scratch Suitable for Algorithmic Thinking Development. Není zadáno v OBD, čeká se na indexaci.

*Publikační výstup (časopis) – časopis indexován v databázi*

[4] Coufal, P., Hornik, T., Hubalovsky, S., Musilek, M., Development and Programming of KarelNXT Robot as a Simulation of xKarel Programming Language Including a Sample Program. *International Journal of Mathematical and Computational Methods. 2017, Vol. 2, p.* 327-331. ISSN: 2367-895X. Do OBD bude zadáno.

[5] Hornik, T., Coufal, P., Musilek, M.,Hubalovsky, S., A Solution of the Mastermind Board Game in Scratch Suitable for Education - Results of the Preliminary Case Study. *International Journal of Computers*. 2017, Vol. 2, p. 214-219. ISSN: 2367-8895. Do OBD bude zadáno.

**Povinné přílohy**

a) výpis (export) z OBD – výsledky publikační činnosti podpořené projektem

b) kopie publikačních výstupů

c) vyúčtování dotace – „Výsledovka po účtech s pohyby“ z ekonomického informačního systému Magion

Datum: 4.1. 2017

Podpis odpovědného řešitele
**Mgr. Petr Coufal**

[illegible]

## Seznam literatury podle šablony ID záznamu

[1]**Coufal, P., Hornik, T., Hubálovský, Š., Musílek, M.** Simulation of the Automatic Parking Assist System as a Method of the Algorithm Development Thinking. *International journal of education and information technologies*. North atlantic university union, 2017. 7s. ISSN: 2074-1316. Kód RIV: AM - Pedagogika a školství.
granty: 0
Spec. výzkum: S.
Forma: J_ČLÁNEK V ODBORNÉM PERIODIKU
(ID: 43873514) (RIV ID: 50013905)

# Simulation of the Automatic Parking Assist System as a Method of the Algorithm Development Thinking

PETR COUFAL, TOMAS HORNIK, STEPAN HUBALOVSKY, MICHAL MUSILEK

*Abstract* - The teaching of algorithm development and programming should develop not only practical skills, theoretical knowledge and techniques, but algorithmic thinking in particular. [1] The ability of algorithmic thinking can be cultivated through implementation of modeling and computer simulation of real phenomena into the teaching of algorithm development not only at universities, but also at high schools and elementary schools. [2, 3] The process of complex problem solving leads the students to implement problem analysis in order to figure out the given problem. Whole process of the problem analysis, including its solution, strategy, algorithm development and creation of the computer simulation, is presented in the article in a step by step form of the case study simulating a real process - the Automatic Active Parking Assist System. Computer simulation of this process is presented by means of robotic system LEGO® Mindstorms® and programmed in the integrated development environment of LEGO® Mindstorms® NXT-G.

*Keywords* - Active Parking Assist, Algorithm Development, Algorithmic Thinking, LEGO® Mindstorms®, Robot Model, Sensors.

## I. Introduction

The ever rising complexity of various educational aids, such as for example LEGO® Mindstorms®, allows teachers to easily include different intricate real-world systems into the common curriculum. There is no longer need for a purely theoretical explanation, that can be substituted by working knowledge of the given system.

Enormous progress has been made in the area of autonomous cars, mostly because of Google self-driving car project, which started in 2009 and was renamed to Waymo in December, 2016. [4] One of the most important parts is the company modification of already existing LiDAR technology Light Detection and Ranging), that is mounted on the car and provides the system with data necessary for making all the decision regarding the driving.

LiDAR itself is a system of lasers measuring ranges from obstacles, which creates precise 3D model of the given area. On top of that Google/Waymo created algorithms that discern exact type of obstacle (pedestrians, cyclists, animals, garbage cans, other cars, buses, and so on) in real time and give the information to the driving program.

Autonomous cars are the future of all the car industry, and as such it is important for people to at least broadly understand the technology behind it. Simulation of a system so complex in school conditions and on a budget is near impossible. However, there is a direct predecessor called Active Parking System or Active Park Assist.

Active Parking Assist also takes control of the whole car, yet it happens only during a parking process. When the driver wants to park and the circumstances are assessed as optimal, the car is able to execute parallel parking all on its own. This system is regularly used by companies like Ford or Mercedes. [5], [6]

In this paper we introduce a case study illustrating both construction and programming of a LEGO® Mindstorms® model, including problem solution, and algorithm development of the Active Parking Assist system. Computer simulation of this process is presented in visual programming system for LEGO® Mindstorms® NXT-G 2.0.

## II. Problem Formulation

The article shows the possibility of a simulation of a real-world system within the confines of a school class. The creation of similarly complex tasks, that links the World of Work (WoW) with the Inquiry-Based Learning (IBL), has been traditional at our department. One of the authors was a member of the international team in MaSciL project [7] oriented on connecting WoW and IBL. To imitate the system, specific problems have to be solved regarding the features and the limitations resulting from the use of LEGO® Mindstorms® construction set:

- What are the minimal necessary dimensions of a parking place for the given car model?
- What is the specific construction of the car?
- Which sensors will be mounted on the car?
- What are the sensors limitations?
- Which algorithm will control the car?

All these questions are connected, because the choice of the car construction defines the minimal dimensions as well as the positioning of the selected sensors. These decisions determine exact form of the final algorithm.

### *Parallel Parking with Active Parking Assist*

In a car with the Active Parking Assist the driver is in full control of the car until he or she reaches a parking lot. Then there are several different versions of Active Parking Assist implementation based on the given car manufacturing company as well as particular car model. The versions vary in the level of car autonomy and the amount of driver's involvement in the parking process.

Our version emulates one of the most advanced variants - the driver completely hands control over to the Active Parking Assist, which continues moving forward while using side ultrasound sensor to determine whether there is necessary amount of space for parallel parking. [8] If such a place is found, the car itself starts the parallel parking maneuver, as depicted in Figure 1.

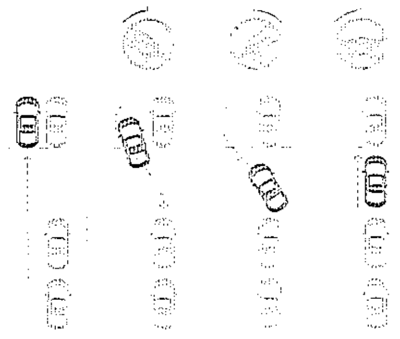
Fig. 1 Parallel parking scheme [8]

## III. Problem Solution

This section demonstrates the actual construction, analysis of individual problems and possible states during the parking process and the algorithm itself. The section is concluded by the flowchart and the program created in the development environment LEGO® Mindstorms® NXT-G 2.0.

### A. *Construction of the car model*

The underlying construction of the robotic car model is based on the original instructions for the "Intelligent Car" created by LEGO® company. The model is from the second generation of LEGO® Mindstorms® referred to as NXT (standing for NeXT) and two construction sets are necessary for its assembly, 9695 and 9797. [9] The basic design was modified and enhanced in order to meet the requirements placed upon the robotic car model.

The requirements include:

- addition of ultrasonic sensor facing the right side of the car model;
- addition of the EOPD sensor (Electro Optical Proximity Detector) facing the right side of the car model;
- addition of light bulbs as signaling devices for reverse car movement;
- efficiency improvement of gears used for model steering;
- maintaining the model compactness.

The resulting construction complies with all the requirements.

There were several different reasons for choosing specifically the "Intelligent Car" model. The design uses only basic construction sets 9695 and 9797. The model utilizes wide range of sensors - an ultrasonic sensor, a light sensor and two touch sensors. The interactive servo motors are located in the central section near the center of gravity. One of the motors controls direct power rack and pinion steering utilizing rubber band transfer, while the other motor procures model movement via rear differential driving axle. Whole model is compact with easy access to the control unit.

### B. *Enhancements made on the construction*

The first model improvement is the replacement of a rubber band transfer located in the steering section. The rubber band was replaced by a set of two cog gears while keeping the same transfer rate 1:1, but changing the resulting rotation direction of the whole steering system. This issue is addressed by a simple change of the motor rotation direction in the control program.

Petr Coufal, Tomas Hornik,stepan Hubalovsky, Michal Musilek are with the Department of Cybernetics, University of Hradec Kralove Rokitanskeho 62, Hradec Kralove, CZECH REPUBLIC
petr.coufal@uhk.cz, tomas.hornik@uhk.cz stepan.hubalovsky@uhk.cz, michal.musilek@uhk.cz

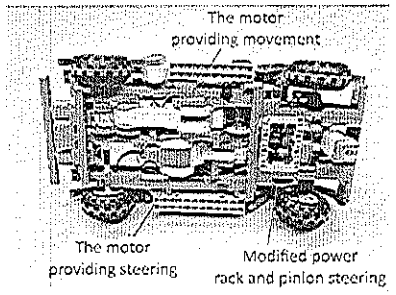

Fig. 2 Model chassis with the modification

Another improvement is located on the right side of the model. Two sensors were added there pointing perpendicularly to the car model. In the back section between the cab and the rear right wheel there is an ultrasonic sensor for making distance measurements. The EOPD sensor incorporated into the front section behind the front right wheel is used for exact measurement of shorter distances. Positioning of sensors in these places is convenient for easy access to input ports of the control unit as well as for leaving the truck deck empty for further expansions of the model. These positions are also particularly suitable for distance measuring needed for the Active Park Assist.

Reverse car movement is signalized by a pair of mutually connected light bulbs placed in the rear section of the truck deck. The bulbs are placed on the upper side of the deck for better visibility, however in case of necessity, they can be moved to the lower side leaving the whole truck deck empty. Reverse car movement signalization has obviously no impact on the functionality of the Active Parking Assist system, but it is an element strengthening the connection to the real world, i.e. WoW.

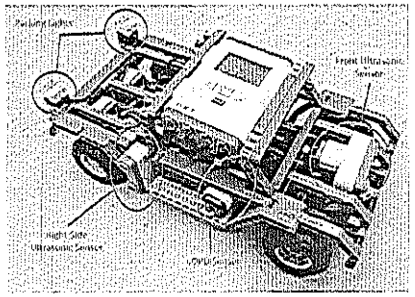

Fig. 3 Enhancements made on the model

The compactness of the model remained intact even after so many changes and enhancements. Keeping the truck deck empty and thus leaving the option for further model expansions was considered as one of the main goals.

## C. *Sensors*

The total number of utilized sensors is six, which is more than the control unit can link. Thus our model uses only selected sensors. In the rear section there are two touch sensors, which can be used for impact monitoring. In the front section there is a light sensor located so that it can follow the surface on which the model is moving. This sensor can be used for tracing and following a black line. None of those sensors was used in the program and remained in the model as a possibility for future expansion.

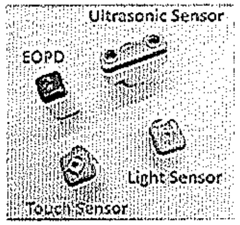

Fig. 4 Different types of sensors used in the construction

Ultrasonic sensor located in the front section measures the distance in front of the model. Analogously the ultrasonic sensor located on the right side along with the EOPD sensor are both used for measuring the distance from the side of the car.

Incorporated NXT motors can be used as drive units or as rotation sensors. This feature is used in our program for the motor propelling the rear differential driving axle.

### *Ultrasonic Sensor*

The ultrasonic sensor is based on sonar principle and serves for measuring the distance up to 250 cm. The measuring is done by means of an integrated source of ultrasonic waves, which sends the waves towards the observed item. The waves bounce back from the item and return back to the sensor that measures the distance based on the time necessary for the whole process.

The distance values are within the interval of 0-100 inches or 0-250 cm. The measurement deviation is ± 3 cm. Measurement quality is dependent on the distance, size and material of the item, that reflects the waves. Big items from hard materials reflect the waves considerably better than small items from soft materials.

### *EOPD Sensor*

Electro Optical Proximity Detector (EOPD) serves for exact measurements of short distances up to 20 cm. Measuring is done by an integrated source of pulse modulated light, which emits light towards the observed item. The sensor records reflected light intensity value, which it compares with established value of background light intensity, which is in our case the place where the measuring is being done.

The sensor makes 350-400 measurements per second. It works in two different modes labeled as 1x and 4x, which differ in measuring distances and in possibilities of item detection. The item detection accuracy depends on its distance, size, shape and above all the surface structure that reflects the light back to the sensor.

Unlike the ultrasonic sensor, the EOPD sensor doesn't work with the time necessary for the return of reflected waves, but with the intensity of reflected light. The raw readings range from 0 to 1024, measurement deviation is 12. From these readings the exact distance is calculated by the following formula:

$$distance = \frac{kScale}{\sqrt{rawsensorvalue}} - kError$$

Fig. 5 EOPD Distance Calculation Formula [10]

In the formula kScale is a constant that sets the scale of the value and kError is a constant defining the difference between the calculated distance and the actual distance. [10] Quoted values are only illustrative and the sensor has to be calibrated for accurate readings and calculations.

### *Rotation Sensor*

NXT motors can be used as a source of rotation movement or as a rotation sensor. Motors include integrated rotation sensor and a sequence of gears, that have resulting gear ration 1:48. The turn of the motor can be regulated in degrees, rotations or in seconds. While using the motor as a rotation sensor the output values are stated either in degrees or in rotations. Positive and negative signs indicate the rotation direction.

Other motor parameters, such as rotational mechanical force, torque moment and rotation speed, are directly dependent on battery voltage powering the control unit.

## D. *Parameters of the Employed Sensors*

The NXT control unit has four input ports for sensors, which is a limiting factor for our model incorporating six sensors in total. In order to plug in all the built-in sensors, or to add other sensors, it is necessary to supplement the model with a sensor multiplexor powered by a separate power source.

Three motor ports on the control unit are filled with a motor for movement, a motor providing the steering and the third port contains the light bulbs for signaling the reverse car movement. In order to extend the amount of motor ports it is again necessary to add a multiplexor for NXT motors.

## E. *Possible Future Expansions of the Model*

The model design can be supplemented with more sensors, which would allow precise parking and also expand the amount of different parking methods. In order to do so, it is necessary to add ultrasound and EOPD sensors also on the left side of the model as well as on the rear side. Such an expansion however requires use of the aforementioned multiplexors.

Another option for further expansion of the model is to use different sensors altogether, e.g. an accelerometer, a compass or a gyroscope.

Apart from the model expansion through addition of more sensors, the option to fully utilize the sensors already mounted on the vehicle by means of the control program enhancement remains. The light sensor situated in the front section of the vehicle model allows the car to follow a black line on the surface, which can be used for fully autonomous driving in a given area and subsequent parking chosen according to the available space.

## IV. ALGORITHM DEVELOPMENT

An algorithm can be defined as a sequence of instructions describing the procedure of solving given problem. The generally accepted norm for algorithm presentation is a flowchart. When the problem is analyzed and the algorithm is created, it can usually be written in more than one programming language. LEGO® created its own integrated development environment called LEGO® Mindstorms® NXT-G 2.0. In following subchapters, we will describe individual problems and solutions.

## A. *Analysis of the possible states*

Fully autonomous system has to be able to deal with all the possible exceptions, that can happen during the course of its execution. For the educational purposes the solution has been simplified and certain exceptions have been left out.

The distance measuring in our model remained very close to the original, however the parking process itself is reduced into precisely given steps that are repeated exactly in the same way. In other words, where the real car uses external sensors for precise parking, our model parks always in the same way, not taking into account any sensor data. This leaves the option for future alteration of the program with more complex parking procedure utilizing aforementioned addition of a sensor multiplexor.

*The starting point*

The beginning of the program execution is created just like the real Active Parking Assist, i.e. the driver arrives at the parking lot, sets the car along the line of already parked cars and then gives over the control to the computer. In our case the model is set at the starting point manually and then the program is started.

*Identification of an obstacle in front of the car*

While searching for an empty parking slot, the car has to continually check, whether there is still space ahead. In reality, the car has to check either for a wall, a curb or some other small obstacle that would prevent the parking process. Our model expects, that the parking lot ends with a wall. With the use of an input port multiplexor, the front light sensor could be checking for a line imitating curb. In case of an obstacle in front of the car in a distance of 35 cm or less, the car stops.

*Measuring the depth and length of a parking space*

The car continually checks with the ultrasonic sensor (or EOPD) distance from the nearest obstacle on the right side. If there is none or if it is further than 20 cm, it means that the car has enough depth to be parked there. From this point, the motor starts behaving also as a rotation sensor counting the length of the parking space. If the space is long enough, the car stops moving forward along the line of parked cars. If there is an obstacle and the distance on the right side of the model becomes smaller than 20 cm, the variable counting the length of the parking space is zeroed.

*Parallel parking maneuver*

If a parking space big enough for the parking maneuver is found, the car parks itself. The real systems are still using the external sensors to check distances on all sides of the vehicle during the whole parking process. Our simplified version jumps out of all the loops checking distances and gives control only to the motors. They are procedurally synchronized to park the car in the same way as shown in figure 1.

## B. *Analysis of the robot motion*

*Move forward*

detection:
- front ultrasonic sensor detects no obstacles in front of the car within the distance of 35 cm;

action:
- the model keeps steering motor centered, while the propulsion motor moves the car forward;
- the side sensor is activated and measures the depth and length of potential parking spaces.

*Start counting the length of a potential parking slot*

detection:
- front ultrasonic sensor detects no obstacles within the 35 cm ahead of the vehicle;
- side ultrasonic sensor detects no obstacles within the 20 cm from the side of the vehicle;

action:
- the propulsion motor begins to serve also as a rotation sensor storing the parking spot length value in a variable.

*Parking slot is too short*

detection:
- front ultrasonic sensor detects no obstacles within the 35 cm ahead of the vehicle;
- side ultrasonic sensor starts detecting an obstacle within the 20 cm from the side;

action:
- the variable holding the parking spot length is zeroed and searching for the beginning of a new parking slot starts again.

*Parallel parking*

detection:
- a parking slot longer than 35 cm is found;

action:
- front and side ultrasonic sensors are stopped;
- the motor stops acting as a rotation sensor;
- sequence of parking instructions is commenced.

*Stop the car, terminate the program*

detection:
- no appropriate parking slot detected;
- the front ultrasonic sensor detects an obstacle ahead of the car in a distance of 35 cm or less;

action:
- front and side ultrasonic sensors are stopped;
- the motor stops acting as a rotation sensor;
- stop the forward movement of the car;
- write error message "No parking space".

## C. *Algorithm of solution*

The solution of the problem is presented in the form of a flowchart (shown on figure 6) as well as a screenshot of the program itself created in the integrated development environment of LEGO® Mindstorms® NXT-G 2.0 (figure 7).

Despite the fact, that it is possible to rewrite the final algorithm in a form of a flowchart and the program itself seems to be procedural, in reality the LEGO® Mindstorms® NXT-G 2.0 belongs under the event-driven programming paradigm.

Continuous distance control from the nearest obstacle in front of the car is difficult to represent in a traditional flowchart. The event of achieving the set limit distance from the obstacle in front calls immediate interruption of the program and switch into the program termination part.

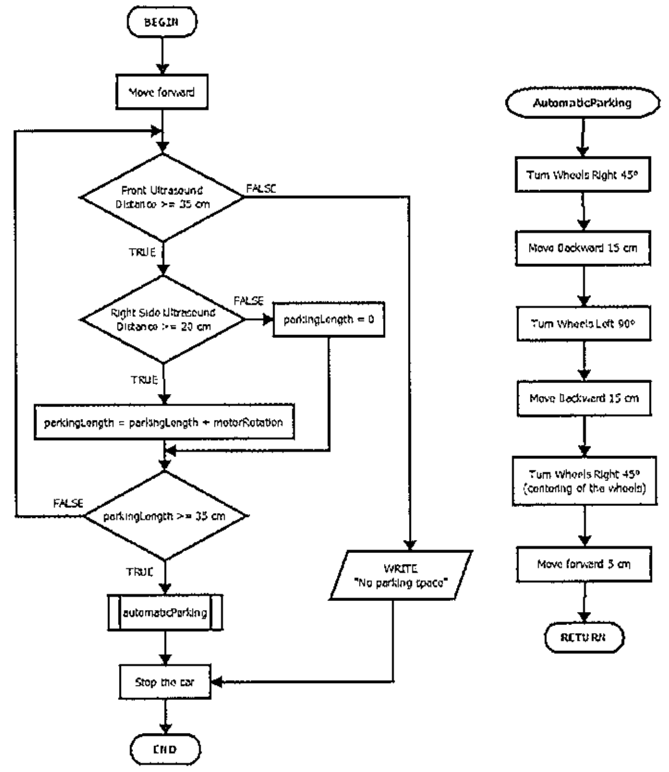

Fig. 6 Flowchart describing the program solution

## V. CONCLUSION

There is a vide variety of different approaches for training and cultivation of algorithm development thinking of pupils and students.

The paper offered insight into one of the possibilities, which is the implementation of computer simulation principles of real world phenomena and processes into the education process of algorithm development and programming. Authors described in detail the problem analysis of an Active Parking System, the problem solution, strategy, algorithm development and creation of the computer simulation within the frame of the case study. Similar examples are incorporated into the programming education at the UHK. [11, 12]

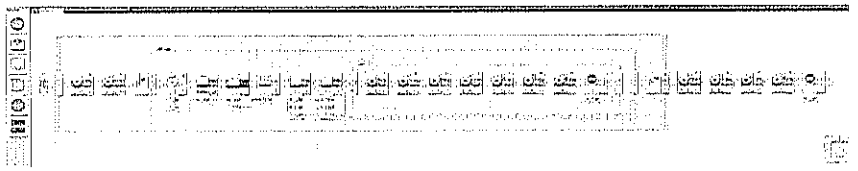

Fig. 7 The NXT-G program for Automatic Parallel Parking

**Acknowledgment**
This research has been partially supported by the Specific research project of the Faculty of Education of the University of Hradec Kralove 2017 and Specific research project of the Faculty of Science of the University of Hradec Kralove 2017.

*References:*

[1] Hubalovsky, S., Modeling and Simulation of Real Process – Passing through the Labyrinth as a Method of Development of Algorithm Thinking and Programming Skills. *International Journal of Mathematics and Computers in Simulation.* 2013, Vol. 7, No. 2, p. 125-133. ISSN 1998-0159.

[2] Hubalovsky, S., Modeling and computer simulation of real process – solution of Mastermind board game. *International Journal of Mathematics and Computers in Simulation.* 2012, Vol. 6, No. 1, p. 107-118. ISSN 1998-0159.

[3] Hubalovska, M., Hubalovsky, S., Implementation of the Systems Approach in Mathematical Modeling, Dynamic Simulation and Visualization Using MS Excel Spreadsheet. *International Journal of Mathematics and Computers in Simulation.* 2013, Vol. 7, No. 2, p. 267-276. ISSN 1998-0159.

[4] Krafcik, John. (Dec 13, 2016). "Say hello to Waymo: What's next for Google's self-driving car project." In: *Medium.* [online]. Accessed March 9, 2017. Available: <https://medium.com/waymo/say-hello-to-waymo-whats-next-for-google-s-self-driving-car-project-b854578b24ee#.26txvbbsg>

[5] Sauser, Brittany. (Jan 6, 2009). "Ford's New Car Parks Itself." In: *MIT Technology Review.* [online]. Accessed March 4, 2017. Available: <https://www.technologyreview.com/s/411563/fords-new-car-parks-itself/>

[6] "Active Parking Assist." In: *Mercedes Benz.* [online]. Accessed March 4, 2017. Available: <http://techcenter.mercedes-benz.com/cs_CZ/active_parking/detail.html>

[7] Michiel, D. et al. 2016. *MaSciL Mathematics and Science in Life: Inquiry Learning and the World of Work.* Freiburg: University of Education. ISBN: 978-90-70786-35-9.

[8] Biello, Rachel. (Jan 1, 2014). "Parallel parking instructions." In: *Behance.* [online]. Accessed March 4, 2017. Available: <https://www.behance.net/gallery/13434875/Parallel-parking-instructions>

[9] The LEGO Group. 2010. "Intelligent Car: 9695 + 9797." In: *Lego Education: Building Instruction.* [online]. Accessed March 6, 2017. Available: <http://lego.sasbadi.com/cms/upload_files/download/g000021.pdf>

[10] "EOPD – How to measure distance." In: *HiTechnic Blog.* [online]. Miami: HiTechnic Products, 2010. Accessed March 6, 2017. Available: <http://www.hitechnic.com/blog/eopd-sensor/eopd-how-to-measure-distance/>

[11] Hubalovsky, S., Musilek, M., Modeling, Simulation and Visualization of Real Processes in LOGO Programming Language as a Method of Development of Algorithm Thinking and Programming Skills. *International Journal of Mathematics and Computers in Simulation.* 2013, Vol. 7, No. 2, p. 144-152. ISSN 1998-0159.

[12] Musílek Michal, Hubálovský Štěpán, Hubálovská Marie. Mathematical Modeling and Computer Simulation of Codes with Variable Bit-Length. *International Journal of Applied Mathematics and Statistics,* Vol. 56; Issue # 1, p. 1-12. ISSN 0973-1377.

# Development and Programming of KarelNXT Robot as a Simulation of xKarel Programming Language Including a Sample Program

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

*Abstract:* - The article is dealing with programming of a KarelNXT robot made from a LEGO construction set. The idea comes from xKarel programming language, which is a traditional programming language utilizing a virtual robot only shown on a screen. The virtual xKarel programming language was extended by its implementation in LEGO Mindstorm NXT-G integrated development environment. Real robots built from LEGO construction sets use additional sensors (compass, color and ultrasonic) which we incorporated in such a way as to make possible building of a robot with equal functions to the robot in xKarel programming language. KarelNXT robot built from LEGO construction set is using the control unit in NXT version. In the article we mention the detailed description of individual robot movement instructions with a description of their meaning and functionality. Finally, we provide a sample program for comparison.

*Key-Words:* - Robot, LEGO, Mindstorms NXT, KarelNXT, xKarel, Sensors, Instructions, Programming Languages

## 1 Introduction
Specific programming languages are addressed in teaching of programming and when shifting to robotics it is necessary to change both the programming language and integrated development environment, which can be very confusing for the elementary school pupils. The issue is often being complicated by the fact that pupils have to learn everything all over again. To make said shift in teaching of programming easier it is convenient to harness the programming similarities. Initial simulation of a robot's movement positively affects the students' experience and makes the work with real robots less problematic [1,2].

Therefore we interconnect the programming language xKarel to the robotic construction set Lego Mindstorms NXT. The aim is to build the robot from Lego construction set using the NXT control unit that is according to our research [3] one of the most widespread units in teaching of programming at elementary and secondary schools in the Czech Republic. Complementary sensors are utilized by built robot Karel NXT in such a way it meets requirements of the robot Karel from program xKarel.

## 2 Construction of KarelNXT robot according to programming language xKarel
For proper construction of KarelNXT robot from Lego Mindstorms NXT construction set is absolutely necessary to understand the way in which robot Karel is moving in xKarel program. Based on the knowledge the requirements for both robot's construction and features are set so that it corresponds to reality. In the control program a set of instructions is created to make the robot controlling analogous to xKarel program.

### 2.1 Programming language xKarel
According to the experts in programming xKarel program is a programming language (programming game) suitable for teaching of structured programming. The program is created in ANSI C++ using FLTK library. It is available at Microsoft Windows®, UNIX® and MacOS® platforms and localized into Czech and English language. [4]

The aforementioned program consists of two windows when the first of them is a room in which robot Karel is moving in square fields.

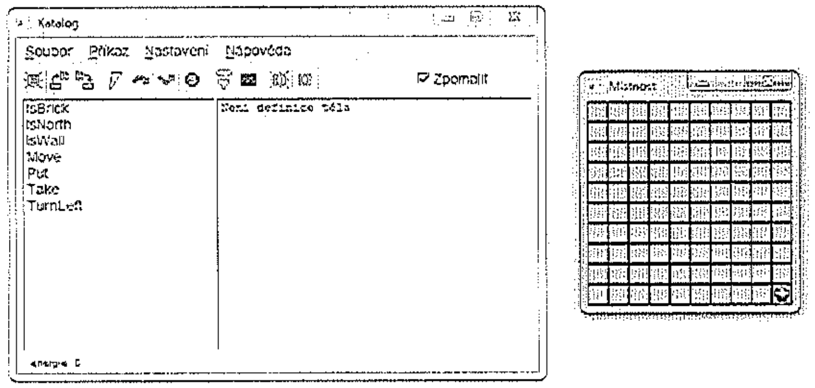

Fig. 1. xKarel development environment

The second window represents a catalogue designated for running the instructions and defining or editing procedures for robot Karel control. The catalogue enables further work with the program.

Robot Karel is situated in a room in which it is moving by means of instructions given to it by the user. The commands for the robot are formed by the instructions in the menu and the user can gradually define and broaden it. Having executed the program, the user selects the basic instructions *Move, Put, Take* and *Turnleft* as well as three conditions *IsNorth, IsWall* and *IsBrick*. For creating of additional instructions (Procedure) is possible to use aforementioned basic instructions and conditions or the cycle *While* and *While Not*. [4]

Basic instructions are indivisible instructions that robot Karel comprehends and other instructions are consisting of.

*Move*: robot Karel makes a step, which means it moves one square forwards in such direction it is heading. If it is situated in front of the wall, the error appears.

*Put*: robot Karel puts down a brick in a square where it is standing. In case of exceeding of the maximal bricks quantity in a field, there occurs the error.

*Take*: robot Karel lifts the brick of the occupied position. When the brick is not in the field, the error emerges.

*Turnleft*: robot Karel turns left 90 degrees.

The conditions are to be used for better robot control. With the help of conditions we can separate the instructions that are and are not to be conducted.

*IsNorth* condition is tracking robot's Karel heading northwards, if the heading is correct the instruction sequence is executed. *IsWall* condition is considered to be true when robot Karel is standing directly in front of the wall. In such a situation the robot cannot make any other step forwards, otherwise it would hit against the wall. *IsBrick*, the third condition, is true if robot Karel is standing in a field where the brick is located. The conditions can be improved by using Else instruction which helps the user to conduct the instruction sequence if the condition is not fulfilled.

In loops we use *While* Condition {the sequence of instructions executed when the condition is fulfilled} or *While Not* Condition {the instruction sequence for not fulfilling the condition}. One of the examples of the sequence using *IsNorth* condition is *ToNorth* instruction that is moving the robot heading northwards before it hits against the wall and terminates the program with the error.

```
procedure ToNorth
 {While IsNorth {Move}
 }
```

Fig. 2. ToNorth program - xKarel

## 2.2 Construction and programming of KarelNXT robot

Robot model built from Lego® construction set has to meet the following requirements in a way it responds robot's Karel behaviour in xKarel program:

- Sensor detecting the wall in front of the robot
- Sensor detecting the brick in a square where the robot is standing
- Sensor determining the cardinal points
- Robot is able to turn 90 degrees on the spot
- Robot is able to inform about potential error on its screen
- Robot is able both to pick and put the bricks together

For meeting the requirements the space representing the room for robot's movement is essential. The criteria are as follows:

- The room is divided into square fields
- The room includes walls
- The room contains bricks

Based on aforementioned requirements KarelNXT robot model was built and room for its movements was designed.

**Robot construction**

KarelNXT robot is built from Lego® construction set using NXT control unit. The construction operates parts of 9797 and 9695 educational kits. In addition there is the compass sensor, the colour sensor and the carboard brick feeder. The robot is using two vertically placed engines for its movement. The bottom part of the robot's chassis operates the rubber strip in three wheels delta arrangement. Strip bottom part of the chassis enables it turns in any direction on the spot. The third engine with the help of the rubber wheel placed on the carboard brick feeder entrance serves as a tool for collecting and piecing the bricks together. Above the robot's engines is NXT control unit. The positioning helps the user to read on-screen information as well as easily manipulate and launch the robot. Another undeniable advantage is the easy availability of all robot's ports. In the raised platform above the control unit can be found the compass sensor used for determining the North. Another sensor is placed in the front-lower part of the robot right in between the engines. The sensor in question is an ultrasonic sensor for the distance measuring used for detection of the walls in the room. In the lower part of the robot, in front of the cardoard brick feeder entrance, is the colour sensor functioning as a brick detector. For accuracy control of the robot's position a gyroscope sensor is often used. Not even that guarantees exact 90 degrees turning of the robot as stated by Dave Riske [5] in case of robot Karel from EV3 construction set.

**Room construction**

Considering the size of the robot a room was created with the help of the duck tape on light straight pad. Length of the square field side is 30 centimetres. From the child construction set blocks are built in a way they represent the walls in the room so that it would be possible to built even irregular rooms. The bricks demonstrate rollers 2.5 cm in diameter and 0.7 cm high with the inclined edge of 0.3 cm in 45 degrees. The bricks are made up of hard wood and painted red. The amount of the bricks in one field is limited by one brick.

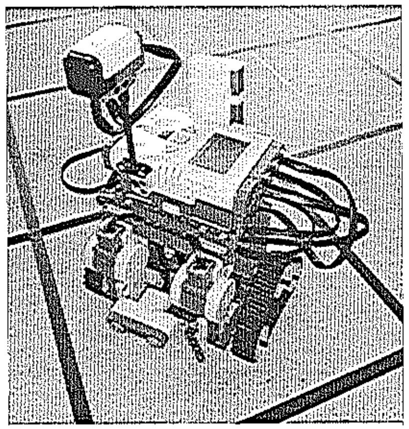
**Fig. 3.** KarelNXT robot construction

**Functions of the sensors**

**The compass sensor**

The compass sensor represents an additional sensor for Lego Mindstorms NXT construction set from HiTechnic company, functioning for robotic models navigation. Digital compass operates with 1° azimuth accuracy, representing values from 0° to 359°, which enables its own definition of the four cardinal points for a room in any direction. Sensor is restoring given value 100 times per second. It is working in two modes – reading and calibration. Compass sensor is susceptible to interfering magnetic field of the engines. Therefore it is located in a raised spot in the robot's construction.

**The colour sensor**

In other words this is an optical sesor making the colour detection of the scan surface much easier. Sensor is able to distinguish six colours (red, blue, green, yellow, red and white) marking them with numbers or selected colour range. The sensor is located in the robot construction for recognizing red bricks situated in a field in front of the robot.

**Ultrasonic sensor**

This sensor is based on the sonar principle and serves for distance measurement in 0-250 cm or 0-100 inches range with ± 3 cm accuracy. The measurment accuracy is influenced by the size, surface, material and the shape of the object which reflects the wave motion back to the sensor. For mor accurate distance measurement the EOPD sensor can be used. [6] The ultrasonic sensor has been found appropriate enough for both robot's KarelNXT construction and its requirements.

**Robot's program**

KarelNXT robot is programmed in the LEGO MINDSTORMS NXT 2.0 program that is a part of the construction set. It is a case of iconographic programming language in which to each active component of the robot the program block is assigned. In the program block the user can set particular parameters of the active robot component. Both simplicity and easy orientation in the program enable its using when teaching in primary schools.

Program makes possible the creating of individual program blocks used during program making for KarelNXT robot. In fact the program block is a subprogram important for setting the final robot operating program.

**Easy instructions:** *Move, Put, Take and Turnleft*

*Move* - In our case *Move* program block is testing right from the beginning whether there is a wall in front of the robot or not. If there is no obstacle, the robot moves one square field forwards, whereas after detecting of the wall there is an audio signal and on the robot's display there appears warning sign "Error, there is a wall.", and the robot stops moving.

*Put* -Put program block is testing if there is a brick in a field. When there is no brick, the robot puts the brick in the field. In such a case a brick is already lying in the field, a sound is heard as well as the warning text "Error, there is a brick!" is displayed on the screen and the robot does not put the brick.

*Take* - From the very beginning Take program block is taking into consideration whether there is a brick in a field or not. If there is a brick in a field, the robot lifts it up. On the other hand there is no brick lying in the field, there is a warning signal and on the robot's screen there appears a text saying "Error, there is no brick!".

*Turnleft* - Turnleft program block is answering for robot's moving in 90 degrees left on the spot.

**Conditions:** *IsNorth, IsWall, IsBrick*

XKarel language syntax offers three conditions altogether used also for KarelNXT robot with the help of program blocks. Given program blocks are functioning as follows:

*IsNorth* program block compares the obtained value from the compass sensor with defined North range (absolute reading 85-95). If the detected value falls within the defined range, the robot is heading north and the first part of the program is being executed. In the opposite situation it is the second part of the program's turn.

*IsWall* program block determines whether a wall is or is not situated in front of the robot. When the distance of the obstacle in front of the robot measured by the ultrasonic sensor is lesser than 20 centimeters, the program block assesses such a situation as a wall obstacle and the first part of the program is executed. After detecting of the distance larger than 20 centimeters there starts the second part of the program for this is a signal meaning there is no wall in front of the robot.

*IsBrick* program block finds out if there is a brick in a field under the robot. Using the colour sensor it compares the colour of the objects located under the robot. When the red colour is detected, the brick is under the robot and the first part of the program starts. When a brick is not in a field under the robot, the second part of the program is executed.

Both the first and the second part of the program is defined by the user in each condition from the program-block menu.

**Cycles**

For better robot control are used cycles operating with aforementioned defined conditions.

*While condition* and *While not condition* - In the first case order sequence is being repeatedly executed for the duration of condition validity whereas during the second one the sequence is being executed as long as the condition is not valid. To make the work with the program more transparent and easier six program block cycles were created:

*WhileIsNorth* program block controls robot's heading to the north, in the positive situation it accomplishes pre-defined order sequence given by the user. Together with the check of robot's heading to the north it is responsible for further program block repetition.

*WhileNotIsNorth* program block represents a negation of *WhileIsNorth* program block.

*WhileIsWall* program block controls if there is a wall in front of the robot. When a wall is actually found there, the program block starts the order sequence having been defined by the user and if necessary it repeats the program block.

*WhileNotIsWall* is a negation of *WhileIsWall* program block.

*WhileIsBrick* program -block controls the position of a brick in a field under the robot and it executes given order sequence. As in other aforementioned cases together with the brick detection in a field under the robot follows with further program block repetititon.

*WhileNotIsBrick* program block is functioning as a negation of *WhileIsBrick* program block.

**Other program blocks**

With the help of all aforementioned program blocks the user is able to create a program for KarelNXT robot as it would be in xKarel programming language. In Lego Mindstorms NXT 2.0 is not possible to insert one program block to another, which means there is no recursive calling as in xKarel program. Now and then this is found to be a disadvantage.

We show two short programs for comparison and illustration.

*TurnRight* (xKarel)

```
Procedure TurnRight
{
Turnleft Turnleft Turnleft
}
```

Fig. 4. TurnRight - xKarel

*TurnRight* (KarelNXT)

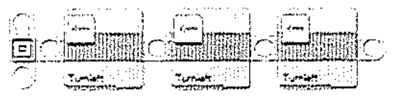

Fig. 5. TurnRight - KarelNXT (Lego Mindstorms NXT)

## 4 Conclusion

Building both KarelNXT robot and its control program block enables us to interconnect computer programming teaching with real model robot programming teaching. KarelNXT robot is built from Lego construction set using additional compass sensor and cardboard for brick feeder. The robot is commanded with given program blocks designed according to xKarel programming language syntax. In Lego Mindstorms NXT 2.0 program a set of program blocks *Move, Put, Take, Turnleft, IsNorth, IsWall, IsBrick, WhileIsNorth, WhileNotIs North, WhileIsWall, WhileNotIsWall, WhileIsBrick, WhileNotIsBrick* was created. Further program blocks are being made by the students during robot programming teaching. We also plan to focus on research and comparison of teaching of programming both in xKarel language and KarelNXT robot.

*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] Klassner, F., Kearney, S.: An Evaluation of Simulation in LEGO Mindstorms Robot Programming Coursework. Las Vegas, CSREA Press, pp.3-9, ISBN: 1-60132-435-9, (2016).

[2] Slangen, L.,Van Keulen, H., Gravemeijer, K.: What pupils can learn from working with robotic direct manipulation environments. International Journal of Technology and Design Education, pp. 449-469, ISSN 0957-7572, (2011).

[3] Coufal, P.: Robotics in Education. Diploma thesis, University of Hradec Kralove, (2016).

[4] XKarel home page [online]. Praha, Robot Karel implementation. Accessed June 20, 2017. Available: http://xkarel.sourceforge.net/eng/. (2017)

[5] Building Karel the Robot: Invaluable Learning Experience for Students [online]. Carson City, Dave Riske. Accessed June 20, 2017. Available: https://nclab.com/building-karel-robot-LEGO/. (2017)

[6] "EOPD – How to measure distance." In: *HiTechnic Blog.* [online]. Miami: HiTechnic Products, 2010. Accessed June 20, 2017. Available: http://www.hitechnic.com/blog/eopd-sensor/eopd-how-to-measure-distance/. (2017)

# 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)

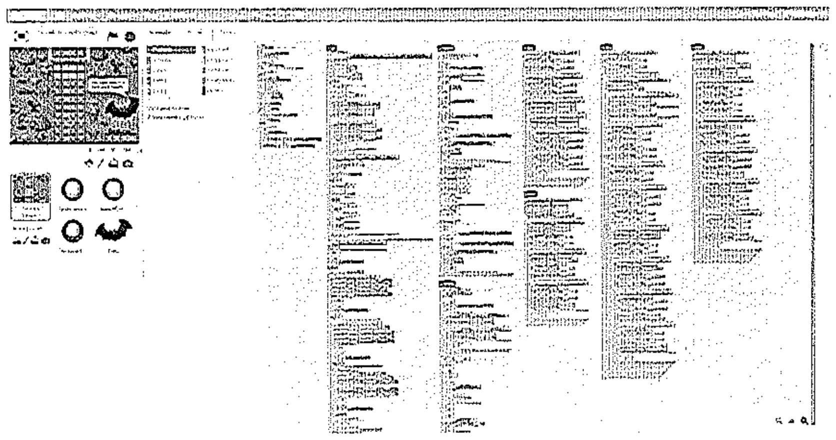

Fig. 1. Overall length of the final program created in Scratch.

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 0W | 0B 1W | 2B 0W | 0B 2W | 1B 1W | | | | |
| 1B | 1B 0W | 1B 1W | 2B 0W | 0B 1W | 0B 2W | 0B 3W | 2B 1W | 1B 2W | 3B 0W |
| 2B | 2B 0W | 0B 2W | 1B 1W | 3B 0W | 0B 3W | 1B 2W | 2B 1W | | |
| 3B | 3B 0W | 2B 0W | 1B 2W | 1B 1W | | | | | |

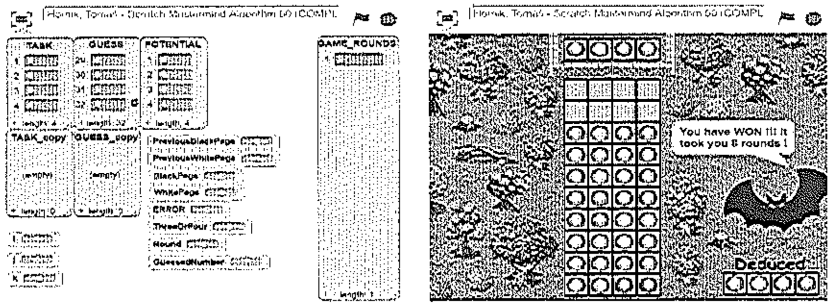

Fig. 2. Side by side comparison of the game. Graphical visualization is on the right side, whereas hidden graphics with most of the variables and lists shown is on the left side.

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] Musilek, 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)