Kuwait J. Sci. **44 (2)** pp. 58-66, 2017

SMAS: A solution-based multi-agent system for improving problem solving skills in computer programming

Danial Hooshyar[1], Rodina Ahmad[2], Mohd H. N. M. Nasir[3*]

Dept. of Software Engineering, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia

[*]*Corresponding author: hairulnizam@um.edu.my*

Abstract

In this research, a solution-based multi-agent system (SMAS) is proposed, which benefits from a novel automatic text-to-flowchart conversion approach in order to improve students' problem solving skills. The aim is to introduce the early stages of learning programming (CS1). By using SMAS, students can focus on solution designing activities in the form of flowchart development more than on language and syntax. Ultimately, an experimental study is devised to assess the success of SMAS as a tool to aid students with problem solving activities and learning computer programming. In total, 30.4% of problems that were left unresolved in previous sessions were solved by students in the control group, whereas 69.7% of previously unresolved problems were solved by students in the experimental group who used SMAS. Therefore, the use of SMAS in practice is supported, as the results indicate considerable gains for the experimental group over the control group.

Keywords: Flowchart; novice programmer; problem solving; solution-based multi-agent system; text-to-flowchart conversion.

1. Introduction and related works

Learning programming is a must for students in the engineering and computer science fields, because several programming courses require programming abilities as part of the curriculum (McCracken *et al.,* 2001). The literature reports high failure and dropout rates in introductory programming courses (Carter *et al.,* 1999). Applying complex programming languages along with a lack of problem-solving skills and solution design activities are outlined by many researchers as the main reasons why learning programming is more difficult (Kartam & Flood, 2002; Pillay & Jugoo, 2005). It is worth mentioning that students' knowledge level, class size, motivation, and programming language syntax are also among the most accentuated reasons. In addition, it is frequently said that the composing instructions for creating program components is another difficulty for many students, which leaves them unable to use basic programming constructs (Spohrer & Soloway, 1985; Deldari *et al.,* 2007; Al-Juboori, 2012). We believe that greater attention and priority should be given to solution designing and problem solving activities, while programming languages should only be used as a way to express solutions. This is because learning to solve problems algorithmically contributes to learning how to program. As majority of students encounter a variety of difficulties in the initial programming stages and may not be able to develop solutions for plain programming problems, they mostly lose interest and even give up. Numerous tools, approaches, and environments have been proposed and developed over the past decades by researchers to overcome learning difficulties faced by students. Some that offer familiar environments to teach basic programming constructs are, for instance, ONTOIAS, an ontology-supported information agent shell for ubiquitous services (Yang, 2011), Jeliot3 (Bednarik *et al.,* 2005) and OOP Semantics System (Teh Noranis, 2011). These help promote students' programming skills in simpler, less complicated environments than professional ones.

Graphical representations have been proposed in numerous animation educational tools, such as SICAS (Chang *et al.,* 2005), Jeliot 2000 (Bassat *et al.,* 2001) and Raptor (Carlisle *et al.,* 2005), to enable students to better understand programs. Several tools have been developed to facilitate the application of artificial intelligence (AI) techniques, such as DISCOVER (Ramadhan *et al.,* 2001) and Lisp Tutor (Anderson & Reiser, 1985) that support individualized learning. Moreover, multi-agents have been proposed to offer students support with problem solving skills. One of them proposed the role of multi-agents that performs a step-by-step extraction and transformation from one problem solving technique to another (Rajabi *et al.,* 2013; Noranis & Azuan, 2013)With these tools, students obtain error and misconception findings as well as corrections for their programs through program simulation. Although it is believed that guiding students to find and correct errors and misconceptions by simulating their own programs is very valuable, students who are weaker cannot benefit from this as they are unable to develop initial solution propositions to be simulated. It is also worth mentioning that instead of focusing on problem solving skills, which are rather essential for weaker students, such tools emphasize more on programming language features. The main concern therefore highlights the need to develop the ability to understand problem description as well as the ability to develop solutions. The main aim of the current research is thus to enhance the problem solving skills of novice programmers. Hence, a Solution-based Multi-agent System (SMAS) is developed, which benefits from an automatic text-to-flowchart conversion approach to enable creating initial solutions for simple problems and to improve problem solving skills. The target audience in this research is novice programmers with no prior knowledge of programming. Many academic applications can also benefit from SMAS and its architecture for problem solving in various areas, drawing diagrams, etc. Moreover, the proposed SMAS may help make teaching programming subjects a more appealing option for instructors. The remaining parts of this article are dedicated to the following topics: SMAS architecture, evaluation and results, and conclusions.

2. SMAS architecture

An illustration of the SMAS architecture is presented in Figure 1. Supporting problem solving abilities through design activities and demonstrating the importance of highlighting the essential principles of various algorithms at higher abstraction levels are considered the primary purposes of SMAS. This system includes two levels, namely “a keyword is found” and “a keyword is NOT found.”

The former occurs when a keyword is found following text processing, such as sentence parsing, noise removal, and the separation of main words, while the latter happens when no keyword is found subsequent to text processing. The main role and function of the first level of SMAS are

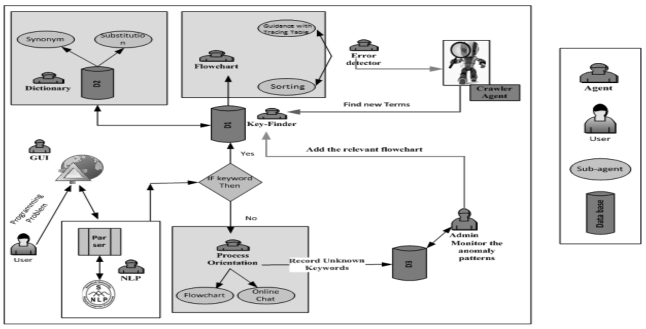


Fig. 1. General SMAS architecture

2.1 The first system level

1.1.1 Graphical user interface (GUI) agent

The interface for the interaction between users and other agents is the GUI agent. Both SMAS levels communicate with this agent as well as the flowchart content and process orientation agent. The GUI agent normally conveys a developed flowchart or sub-flowchart along with instant feedback and messages to users. Moreover, this agent takes the given programming problem, in English text here, and sends it to the next agent called the NLP agent.

1.1.2 Natural language processing (NLP) agent

The NLP agent performs the semantic and syntactic analysis of a programming problem entered in English text. It normally does sentence segmentation, part-of-speech tagging and parsing. After parsing a sentence, the agent performs noise removal, meaning that prepositions, conjunctions, etc. will be removed and only the main words will be sent to the key finder agent.

Example 1: *Write a program to calculate the factorial of a given number.*

After parsing, we have: *write/VB (verb, base form) a/DT (determiner) program/NN (noun, singular or mass) to/TO (to) calculate/VB (verb, base form) factorial/NN (noun, singular or mass) of/IN (preposition or subordinating conjunction) given/VBN (verb, past participle) number/NN (noun, singular or mass)*

As seen above, an online parser processes the entered text and the system automatically removes noise from the parsed sentence. The output for example 1 after noise removal is:

Calculate/VB (verb, base form) factorial/NN (noun, singular or mass) given/VBN (verb, past participle) number/NN (noun, singular or mass)

1.1.3 Key finder agent

This agent cross-checks the main words extracted from the entered sentence with keywords stored in database 1 (D1). If a word matches a keyword, the keyword will be referred to the flowchart agent for further processing. If no keyword is found, the main words will be sent to the dictionary agent for further checking. If no substitution or synonym is found, the second system level starts working.

1.1.4 Dictionary agent

This agent checks words for their synonyms and substitutions through database 2 (D2). If any keyword is found from its two sub-agents, it will be returned to the key finder agent again for further action. This agent comprises two sub-agents as described below:

- *Synonym sub-agent*

This sub-agent cross-checks words with its repository and if any synonym is found, it will be passed to the dictionary agent.

Example 2: *write a program to pick the largest of a set of numbers*

After parsing, we have: *write/VB (verb, base form) a/DT (determiner) program/NN (noun, singular or mass) to/TO (to) pick/VB (verb, base form) the/DT (determiner) largest/JJS (adjective, superlative) of/IN (preposition or subordinating conjunction) a/DT (determiner) set/NN (noun, singular or mass) of/IN (preposition or subordinating conjunction) numbers/NNS (noun, plural)*

If the key finder agent cannot find any keyword match for its question, it refers it to the dictionary agent, which uses a synonym sub-agent to find synonyms such as biggest, maximum, or max for the main word, 'largest.'

- *Substitution sub-agent*

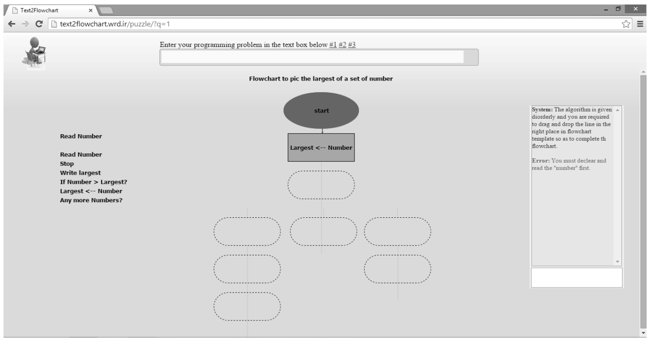
This sub-agent cross-checks the words with its repository and if any substitution is found, it will be passed to the dictionary agent.

1.1.5 Flowchart agent

This agent receives the keyword found from D1 and provides the GUI agent with a workspace for users to complete the sub-flowchart, a system chat to guide users step by step, the flowchart template, etc. System assessment offers a means to guide student learning and feedback for the student regarding the learning process. The flowchart agent includes three sub-agents as follows:

- *Guidance sub-agent*

By employing this sub-agent, users will receive an editor, the algorithm of the programming problem entered, a flowchart template, and a system chat. In this option, users are required to drag the disorderly algorithm given at the left side of the workspace (the lines) and drop it in the right position of the flowchart template. Afterwards, the correct shape will be extracted from the relevant database and be placed through the flowchart. If the algorithm is dropped in the wrong position, the system will not allow the line to be dropped and the system chat will automatically generate the relevant error and feedback. A screen shot of the guidance option after a user drops the algorithm in the wrong position is shown in Figure 2.


Enter your programming problem in the text box below #1 #2 #3

Flowchart to pic the largest of a set of number

Read Number

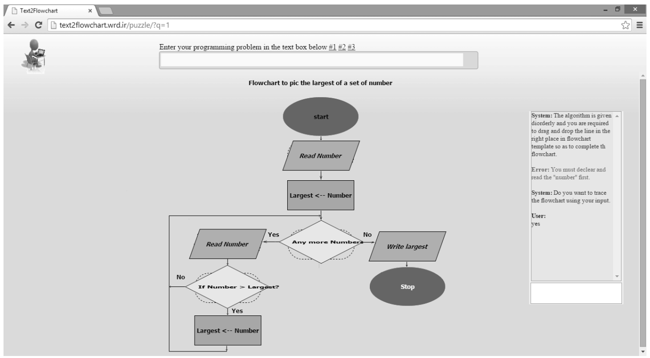
Read Number
Stop
Write largest
If Number > Largest?
Largest <-- Number
Any more Numbers?

System: The algorithm is given diorderly and you are required to drag and drop the line in the right place in flowchart template so as to complete th flowchart.

Error: You must declear and read the "number" first.

Fig. 2. Screen shot of the guidance option after a user drops the algorithm in the wrong position

Once the user has completed the flowchart, the system asks whether they would like to have the complete flowchart traced using a trace table. A snapshot of a typical complete flowchart is shown in Figure 3. If affirmative, the system traces the flowchart for the user to assure they understand the solution of the problem step by step. A snapshot of the generated flowchart with the relevant trace table is shown in Figure 4.


Enter your programming problem in the text box below #1 #2 #3

Flowchart to pic the largest of a set of number

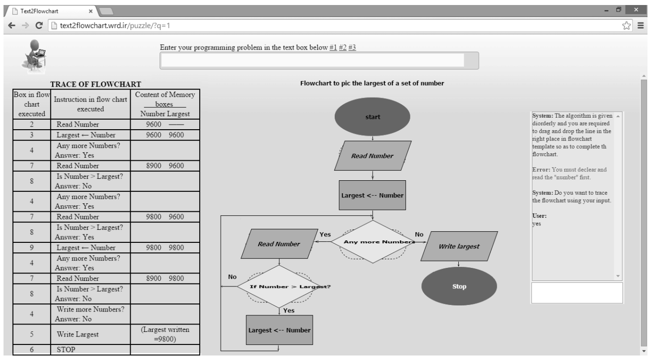
System: The algorithm is given diorderly and you are required to drag and drop the line in the right place in flowchart template so as to complete th flowchart.

Error: You must declear and read the "number" first.

System: Do you want to trace the flowchart using your input.

User:
yes

Fig. 3. Complete flowchart with the guidance option of SMAS


Text2Flowchart

text2flowchart.wrd.ir/puzzle/?q=1

Enter your programming problem in the text box below #1 #2 #3

TRACE OF FLOWCHART

Box in flow chart executed	Instruction in flow chart executed	Content of Memory boxes Number Largest
2	Read Number	9600 ——
3	Largest ⟵ Number	9600 9600
4	Any more Numbers? Answer: Yes	
7	Read Number	8900 9600
8	Is Number > Largest? Answer: No	
4	Any more Numbers? Answer: Yes	
7	Read Number	9800 9600
8	Is Number > Largest? Answer: Yes	
9	Largest ⟵ Number	9800 9800
4	Any more Numbers? Answer: Yes	
7	Read Number	8900 9800
8	Is Number > Largest? Answer: No	
4	Write more Numbers? Answer: No	
5	Write Largest	(Largest written =9800)
6	STOP	

Flowchart to pic the largest of a set of number

System: The algorithm is given diorderly and you are required to drag and drop the line in the right place in flowchart template so as to complete th flowchart.

Error: You must declear and read the "number" first.

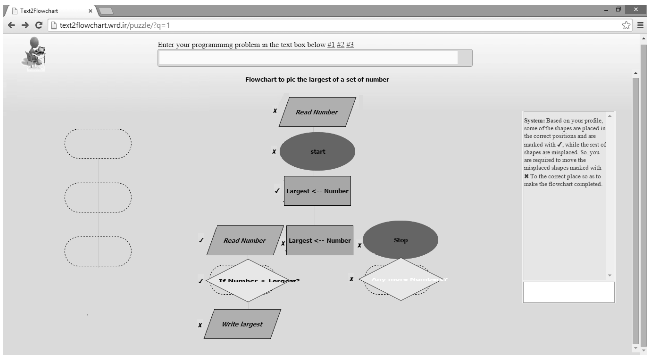
System: Do you want to trace the flowchart using your input.

User:
yes

Fig. 4. Snapshot of generated flowchart and relevant trace table

- *Sorting sub-agent*

Unlike guidance, in the sorting option the algorithm is not provided at the left of the workspace. Some shapes are already placed in the correct places and are marked with √ in the flowchart template and the remaining shapes are misplaced and marked with ×. Therefore, the user is required to sort the misplaced shapes using the empty flowchart template at the left to complete the flowchart. In case the user drags a shape into the wrong position, the same mark (×) will appear next to the shape. In the sorting option, the errors and feedback are brief, as brief feedback is a classical way of making users think after a failure/error. A snapshot of the sorting option is illustrated in Figure 5.


Text2Flowchart

text2flowchart.wrd.ir/puzzle/?q=1

Enter your programming problem in the text box below #1 #2 #3

Flowchart to pic the largest of a set of number

System: Based on your profile, some of the shapes are placed in the correct positions and are marked with ✓, while the rest of shapes are misplaced. So, you are required to move the misplaced shapes marked with ✖ To the correct place so as to make the flowchart completed.

Fig. 5. Snapshot of the sorting option in SMAS

- *System chat agent*

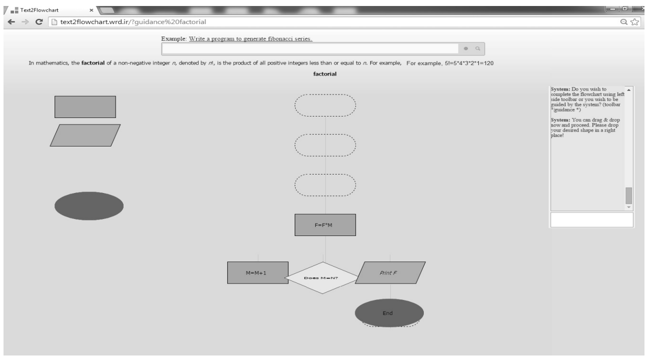
This agent provides users with immediate feedback, errors and recommendations while they are completing the flowchart, as shown in previous figures of SMAS.

1.1.6 Error detection agent

If any error occurs throughout flowchart completion while using the toolbar and guidance sub-agents, the error detection agent will detect and store it. Afterwards, the error will be conveyed to the crawler agent to find additional, relevant information and definitions to automatically improve the database without human intervention for subsequent users.

1.1.7 Crawler agent

This agent receives an unknown keyword from the error detection agent and crawls relevant websites to find a related definition and context for improving the D1 database. Upon extracting additional information by this agent, it will be added automatically to the D1 database. If the next user enters the same question before proceeding to flowchart completion, the system will present this added information in definition form at the top of the page. Figure 6 illustrates the functions of these two agents in example 1.


Text2Flowchart

text2flowchart.wrd.ir/?guidance%20factorial

Example: Write a program to generate fibonacci series.

In mathematics, the **factorial** of a non-negative integer *n*, denoted by *n*!, is the product of all positive integers less than or equal to *n*. For example, For example, 5!=5*4*3*2*1=120

factorial

System: Do you wish to complete the flowchart using left side toolbar or you wish to be guided by the system? (toolbar *|guidance *)

System: You can drag & drop now and proceed. Please drop your desired shape in a right place!

Fig. 6. Workspace of the guidance sub-agent with extra information added by the crawler agent

2.2 The second system level

The role of each agent in the second level regarding 'a keyword is NOT found' is presented.

1.1.1 GUI & NLP agents

These two agents were described in Sections 4.1.1 and 4.1.2.

Example 3: *Write a program that asks the user to type an integer, and write _you win_ if the value is between 56 and 78.*

After parsing, there is*: write/VB (verb, base form) a/DT (determiner) program/NN (noun, singular or mass) that/WDT (Wh-determiner) asks/VBZ (verb, 3rd person singular present) the/DT (determiner) user/NN (noun, singular or mass) to/TO (to) type/VB (verb, base form) an/DT (determiner) integer/NN (noun, singular or mass) and/CC (coordinating conjunction) write/VB (verb, base form) _/VBG (verb, gerund or present participle) you/PRP (personal pronoun) win/VB (verb, base form) _/NNS (noun, plural) if/IN (preposition or subordinating conjunction)* *the/DT (determiner) value/NN (noun, singular or mass) is/VBZ (verb, 3rd person singular present) between/IN (preposition or subordinating conjunction) 56/CD (cardinal number) and/CC (coordinating conjunction) 78/CD (cardinal number)*

As seen above, the text entered is processed by an online parser and the system automatically does minor noise removal in the parsed sentence. It is worth noting that the noise removal stage in the second level of SMAS differs from the first level. Therefore, the output for example 3 after noise removal is:

Asks/VBZ (verb, 3rd person singular present) user/NN (noun, singular or mass) type/VB (verb, base form) integer/NN (noun, singular or mass) write/VB (verb, base form) _/VBG (verb, gerund or present participle) you/PRP (personal pronoun) win/VB (verb, base form) _/NNS (noun, plural) if/IN (preposition or subordinating conjunction) value/NN (noun, singular or mass) is/VBZ (verb, 3rd person singular present) between/IN (preposition or subordinating conjunction) 56/CD (cardinal number) 78/CD (cardinal number)

1.1.2 Key finder agent

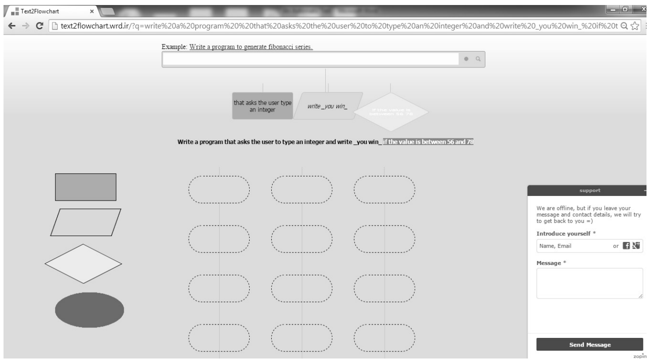
This agent cross-checks the main words extracted from the sentence entered by the NLP agent with keywords stored in D1 and D2. If no match is detected, the main words extracted from the statement of the programming problem will be sent to the process orientation agent.

1.1.3 Process orientation agent

This agent obtains the main words from the NLP agent, refers each related word to its corresponding flowchart notation and then sends them to the flowchart sub-agent for drawing. The process orientation agent includes two sub-agents as follows:

- *Flowchart sub-agent*

This sub-agent refers each main word and keyword to its corresponding shape and develops a sub-flowchart. It also provides the GUI agent with a workspace for users to complete the sub-flowchart, an online system chat to guide users step-wise and a flowchart template. In example 3, the process orientation agent automatically generates the relevant sub-flowcharts using the flowchart sub-agent shown in Figure 7. When users keep the mouse cursor on the programming problem statement, the relevant sub-flowchart will be highlighted to show the relationship between the text and its corresponding flowchart.


text2flowchart.wrd.ir/?q=write%20a%20program%20%20that%20asks%20the%20user%20to%20type%20an%20integer%20and%20write%20_you%20win_%20if%20t

Example: Write a program to generate fibonacci series.

Write a program that asks the user to type an integer and write _you win_ if the value is between 56 and 78

We are offline, but if you leave your message and contact details, we will try to get back to you =)

Introduce yourself *

Name, Email

Message *

Fig. 7. Workspace of the process orientation agent

- *Online chat sub-agent*

It is not unusual for students to get stuck in certain programming assignment stages. In such situation, it is important to get timely help from an instructor to be able to continue working on the assignment. Otherwise, learners may give up or not have sufficient time to work on the task. Therefore, this agent provides novices with an online chat with the system admin for further flowchart development. As shown in Figure 7, an online chat is improvised at the bottom right of the page to help users obtain additional guidance from the system admin. It is worth mentioning that if databases 1 and 2 are fed properly and with sufficient basic exercises aimed at CS minors, users will not be referred to this stage. Therefore, the online chat with the admin is only used in the worst case scenarios.

1.1.4 Admin agent

The admin agent asks the system admin to define and draw the relevant flowchart of each unknown programming problem stored in database 3, which will all be automatically added to D1 to improve the system's database.

3. Evaluation

To assess and investigate the success of SMAS in learning computer programming and acquiring problem solving skills, an experimental study was adopted using 42 first-year undergraduate students taking their first introductory programming course. Because the module content involves imperative strategies for solving fundamental programming problems, these participants were particularly appropriate for investigating SMAS efficacy in learning computer programming and acquiring problem solving skills.

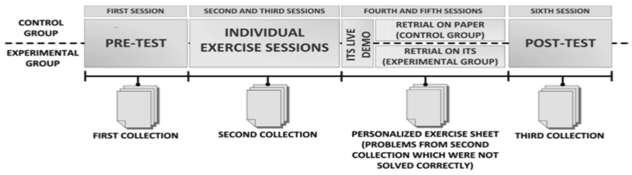

FIRST SESSION
SECOND AND THIRD SESSIONS
FOURTH AND FIFTH SESSIONS
SIXTH SESSION
CONTROL GROUP
EXPERIMENTAL GROUP
PRE-TEST
INDIVIDUAL EXERCISE SESSIONS
ITS LIVE DEMO
RETRIAL ON PAPER (CONTROL GROUP)
RETRIAL ON ITS (EXPERIMENTAL GROUP)
POST-TEST
FIRST COLLECTION
SECOND COLLECTION
PERSONALIZED EXERCISE SHEET (PROBLEMS FROM SECOND COLLECTION WHICH WERE NOT SOLVED CORRECTLY)
THIRD COLLECTION

Fig. 8. Experimental study

The present experimental study was conducted over six sessions in line with a pre-test/post-test control group design. The first and last sessions were 90 minutes each, and the rest were run for 50 minutes each. Three different sets were utilized in the sessions, each including 4 basic programming concepts and problems. The evaluation process is shown in Figure 8. First, the participants were randomly assigned to the experimental and control group. Each group had 21 students. The pre-test conducted in the first session was to test the initial knowledge of the participants using a set of basic programming concepts and problems. In the second and third stages, the participants were given 4 basic programming problems and their answers were collected. Afterwards, the lecturer assessed the answers, based on which a personalized and individualized exercise was handed out to each participant in the fourth session. The problems left unresolved in the two previous sessions were given to the participants as personalized exercises and they spent the fourth and fifth sessions attempting to solve them. In this stage, the experimental group participants only accessed SMAS to solve the problems. A 40-minute live demo was given to the participants with three programming problems in all the sets utilized during the experimental evaluation. Both control and experimental groups solved these three exercises in order to prevent any possible bias. In session six, the third set of programming problems was used to conduct a post-test to measure progress. Scoring was based on algorithm and flowchart (solution design) development; those solutions that contained a correctly developed solution were marked as correct. To assign the pre-test and post-test scores, the number of exercises solved in the first and last sessions was considered. Both homoscedasticity (Levene's test) and normality (K–S normality test) of all variables were checked for evaluation purposes. SPSS 19.0 was used to obtain the mean as a measurement of the central trend and the standard error of the mean (SEM) as a measurement of dispersion. In order to define the influence of using SMAS on the scores gained by the participants (dependent variable), a mixed model ANOVA [group (2; control and experimental) * testing time (2; pre-test and post-test)] was implemented. In addition, in the case of significant interaction and main effects, which was set at p = 0.05 prior to Bonferroni correction, post-hoc analysis with Bonferroni correction was used. Subsequently, the main effect of the test type (pre-test and post-test) on the scores obtained by participants ($F_{1,42} = 20.31$; $p < 0.001$; $\eta^2_p = 0.28$) was displayed by ANOVA. Moreover, testing time and group had an interaction effect on the scores ($F_{1,42} = 7.12$; $p = 0.007$; $\eta^2_p = 0.23$). By reviewing the control group participants' scores in the pre-test and post-test ($F_{1,42} = 2.48$; $p > 0.05$; $\eta^2_p = 0.11$), no considerable improvement was observed. However, unlike the control group, considerable improvement was observed in the experimental group ($F_{1,42} = 27.60$; $p < 0.001$; $\eta^2_p = 0.54$), signifying the substantial influence of the multi-agent system only on the experimental group who utilized SMAS (Table 1).

Table 1. Differences between testing times for the scores gained by the control and experimental groups

	Initial test results	**Final test results**
Control Group (21)	6.56 (1.09)	6.93 (0.55)
Experimental Group (21)	5.66 (0.92)	7.95 (1.30)*

*Considerable differences (P < 0.001) among the scores achieved in the pre and post test

Table 2. Differences in performance upon using the multi-agent system in the treatment stage (sessions 2-5)

	First attempt	**Individualized sheet**	**Relative performance (%)**
Control Group (21)	4.78 (0.69)	1.77 (0.51)	30.4
Experimental Group (21)	5.27 (0.65)	4.83 (0.50)	69.7

The average numbers of exercises solved by the control and experimental groups in the first two sessions are shown in Table 2. The similar prior knowledge of participants in both groups is evident from the small differences in scores gained in sessions 2 and 3 as their first attempt. However, the considerable differences in scores gained in sessions 4 and 5 (with the personalized exercises and sheets) are indicative of the positive and significant influence of SMAS. In the control group, only 30.4% of problems that were left unresolved in previous sessions were solved by students; however, 69.7% of previously unresolved problems were solved by students in the experimental group who used SMAS. The use of SMAS in practice is supported by the results that indicate considerable gains for the experimental group over the control group. The constructive impact of using programming environments aimed at creating a mental model of the solution to a problem for novice programmers as determined in the current study is in line with previous research (Teh Noranis, 2011; Hooshyar *et al.,* 2015, 2016).

4. Conclusion

Literature in this field reports high failure and dropout rates in introductory programming courses. Many students avoid programming in their final year projects because *they do not have sufficient skill in programming*. In this study, a solution-based multi-agent system (SMAS) was proposed and developed. The purpose was to improve the problem solving skills of novice programmers by providing them with algorithm and flowchart development for imperative, basic programming problems. When using this tool, learners are engaged in developing flowcharts by employing an automatic text-to-flowchart conversion approach. SMAS offers students step-by-step guidance to develop flowcharts in the form of solution designing along with, additional information regarding the programming problem entered, and feedback on their actions. In order to assess and investigate the success of SMAS in learning computer programming and acquiring problem solving skills, an experimental study was adopted using 42 first-year undergraduate students, who were taking their first introductory programming course at a public university in Malaysia. In the control group, only 30.4% of problems that were left unresolved in previous sessions were solved by students, but 69.7% of previously unresolved problems were solved by students in the experimental group who used SMAS. The practical use of SMAS is supported by the results, which indicate considerable gains for the experimental group over the control group.

5. Acknowledgment

This work is financially supported by University of Malaya under project RG327-15AFR.

References

Al-Juboori H.M. (2012). Design of an interactive platform for computer engineering education. Kuwait Journal Science, **39** (2A):119-130.

Anderson, J. & Reiser, B. (1985). The LISP Tutor, Byte, **10**:159-175.

Bednarik, R., Joy, M., Moreno, A., Myller, N. & Sutinen, E. (2005). Multi agent educational system for program visualization. Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05); Pp.1-6.

Ben-Bassat, R., Ben Ari, M. & Uronen P. (2001). An extended experiment with Jeliot 2000, In Proceedings of the First International Program Visualization Workshop, University of Joensuu Press, Porvoo Finland, 131-140.

Carlisle, M.C., Wilson, T., Humphries J. & Hadfield S. (2005). RAPTOR: A visual programming environment for teaching algorithmic problem solving, Proceeding of 36th SIGCSE Technical Symposium on Computer Science Education, 176-180.

Carter J. & Jenkins T. (1999). Gender and programming: What's going on? Proceeding of 4th Annual Conference on Innovation and Technology in Computer Science Education, 1-4.

Chang, K.E., So, Y.T. & Lin, H.F. (2006). Computer-assisted learning for mathematical problem solving, Computers & Education, **46**:140–151.

Deldari, H., Sabeghi, M. & Mafi, R. (2007). An agent-based approch to grid programming. Kuwait Journal Science, **34**(2):145-164.

Hooshyar, D., Rodina, B.A., Yousefi, M., Fathi, M., Horng, S.J. & Lim, H. (2016). Applying an online game-based formative assessment in a flowchart-based intelligent tutoring system for improving problem-solving skills, Computers & Education, **94**:18-36.

Hooshyar, D., Rodina, B.A., Raj R.G., Mohd Hairul, N.M.N., Yousefi, M., Horng, S.J. & Rugelj, J. (2015). A flowchart-based multi-agent system for assisting novice programmers with problem-solving activities, Malaysian Journal of Computer Science, **28**(2):133-151.

Kartam, N. & Flood, I. (2002). Enhancing engineering education using multimedia web-based techniques. Kuwait Journal Science, **29** (2):181-196.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagan, D., Kolikant, Y.B., Laxer, C., Thomas, L., Utting, I. & Wilusz, T. (2001). A multinational, multi-institutional study of assessment of programming skills of first-year CS students, in Proceedings of 6th Annual Conference on Innovation and Technology in Computer Science Education, Canterbury, UK, 125-180.

Pillay, N. & Jugoo, V. (2005). An investigation into student characteristics affecting Novice programming performance, ACM SIGCSE Bulletin, **37**:107-110.

Rajabi, M., Teh Noranis, M.A. & Sulaiman M.N. (2013). Computational problem solving architectural design based on multi-agent, Journal of Theoretical and Applied Information Technology (JATIT), **58**(2):311-318.

Ramadhan, H.A., Deek, F. & Shihab, K. (2001). Incorporating software visualization in the design of intelligent diagnosis systems for user programming, Artificial Intelligence Review, **16**(1):61-84.

Spohrer, J.C. & Soloway, E. (1985). Putting it all together is hard for novice programmers, in Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, Tucson, Arizona, 728-735.

Teh Noranis, M.A. & Azuan S.N. (2013). A multi-agent model for information processing in computational problem solving, International Journal of Modeling and Optimization, **3**(6):490-494.

Teh Noranis, M.A. (2011). Object-oriented programming semantics representation utilizing agents, Journal of Theoretical and Applied Information Technology (JATIT), **32**(1): 88-98.

Wheeler, J.L. & Regian, J.W. (1999). The use of a cognitive tutoring system in the improvement of the abstract reasoning component of word problem solving, Computer in Human Behavior, **15**(2):243–254.

Yang, S.Y. (2011). OntoIAS: An ontology-supported information agent shell for ubiquitous services. Expert Systems with Applications, **38**: 7803-7816.

Submitted **:** 30/06/2015
Revised **:** 03/05/2016
Accepted **:** 03/05/2016

نظام قائم على الحل متعدد الوسائط لتحسين مهارات حل المشكلات في برمجة الكمبيوتر

دانيال هوشيار، رودينا أحمد، محمد حريول نظام ناصر*
قسم هندسة البرمجيات، كلية علوم الحاسوب وتقنية المعلومات، جامعة مالايا، كوالالمبور، ماليزيا
*hairulnizam@um.edu.my

خلاصة

في هذا البحث، تم اقتراح نظام قائم على الحل متعدد الوسائط (SMAS)، الذي يستفيد من نهج تلقائي جديد لتحويل النص إلى مخطط لتحسين مهارات حل المشكلات لدى الطلاب. والهدف من ذلك هو تقديم المراحل الأولى من تعلم البرمجيات (CS1). باستخدام SAMS، يستطيع الطلاب التركيز على أنشطة تصميم الحلول على هيئة تطوير المخططات أكثر من التركيز على اللغة وبناء الجملة. وأخيراً، تم وضع دراسة تجريبية لتقييم نجاح SAMS كأداة لمساعدة الطلاب في أنشطة حل المشكلات وتعلم برمجة الحاسوب. إجمالياً، تم حل 30.4٪ من المشكلات التي لم يتم حلها في الدورات السابقة من قبل طلاب مجموعة التحكم، في حين تم حل 69.7٪ من المشاكل التي لم يتم حلها سابقاً من قبل الطلاب في المجموعة التجريبية الذين استخدموا SMAS. لذلك، تم دعم استخدام SMAS في الممارسة العملية، حيث تشير النتائج إلى تحقيق مكاسب كبيرة لصالح المجموعة التجريبية على المجموعة الضابطة.