A FLOWCHART-BASED MULTI-AGENT SYSTEM FOR ASSISTING NOVICE PROGRAMMERS WITH PROBLEM SOLVING ACTIVITIES

Danial Hooshyar [1], Rodina Binti Ahmad[2], Ram Gopal Raj[3], Mohd Hairul Nizam Md Nasir[4], Moslem Yousefi[5], Shi-Jinn Horng[6], and Jože Rugelj[7]

[1, 2, 3, 4] Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur, Malaysia
[5]Center of Systems and Machines Intelligence, College of Engineering, Universiti Tenaga Nasional, Kajang, Malaysia
[6]Department of Computer Science and Information Engineering, National Taiwan University of Science and Technology, Taiwan
[7]Faculty of Education, University of Ljubljana, Ljubljana, Slovenia

E-mail:[1]Danial.hooshyar@gmail.com, [2]rodina@um.edu.my, [3]ramdr@um.edu.my, [4]hairulnizam@um.edu.my, [5]moslem_yousefi@yahoo.com, [6]horngsj@yahoo.com.tw, [7]joze.rugelj@pef.uni-lj.si

Abstract

In the early stages of learning computer programming, Computer Science (CS) minors share a misconception of what programming is. In order to address this problem, FMAS, a flowchart-based multi-agent system is developed to familiarize students who have no prior knowledge of programming, with the initial stages in learning programming. The aim is to improve students' problem solving skills and to introduce them to the basic programming algorithms prior to surface structure, using an automatic text-to-flowchart conversion approach. Therefore, students can focus less on language and syntax and more on designing solutions through flowchart development. The way text-to-flowchart conversion as a visualization-based approach is employed in FMAS to engage students in flowchart development for subsequent programming stages is discussed in this paper. Finally, an experimental study is devised to assess the success of FMAS, and positive feedback is achieved. Therefore, using FMAS in practice is supported, as the results indicate considerable gains for the experimental group over the control group. The results also show that an automatic text-to-flowchart conversion approach applied in FMAS successfully motivated nearly all participants in problem solving activities. Consequently, the results suggest additional, future development of our proposed approach in the form of an Intelligent Tutoring System (ITS) to make the early stages of learning programming more encouraging for students.

Keywords: Flowchart; Novice programmers; Text-to-flowchart Conversion; Problem solving; Visualization

1.0 INTRODUCTION AND LITERATURE REVIEW

Introductory programming learning causes difficulties to many students worldwide. Since there are several programming courses in fields like engineering and computer science, students in these fields should be able to do programming [1]. High dropout and failure rates in initial programming courses are reported in literature [2]. For instance, as Carter and Jenkins indicated, in final year projects students mostly avoid programming because they are not able to program or do not believe they can [3]. The reason for these difficulties is the lack of problem solving skills, solution designing, and the use of programming languages that are often artificial [4-7]. However, students' background in science, motivation, class size, and programming language syntax are additionally highlighted as reasons for this difficulty. Commonly, the basic programming constructs are understood by the majority of students; but they are still unable to employ them for creating programs to solve problems. Due to the aforementioned reasons, to coordinate and compose instruction for a program is a major issue for many students [8-9].

We believe programming languages are merely a way of expressing solutions while more focus should be directed toward problem solving abilities, since learning to solve problems algorithmically contributes to learning how to program. As programming skills cannot be completely transferred from instructors to students, programming should also be actively practiced by novice programmers to gain knowledge [10-11]. However, countless students are unable to develop solutions for simple problems and encounter difficulties in the preliminary learning stages. This might cause loss of interest and giving up, which lead to dropout and failure.

Numerous tools, approaches, and environments have been proposed and developed over the past decades to overcome learning difficulties faced by students [12, 13]. Some offer familiar environments to teach basic programming constructs, for instance micro worlds (e.g., Alice [14, 15] and Karel Robot [16]), which are applicable to movement control and other behaviors of familiar entities. Among the tools proposed, Ruru [17], BlueJ [18, 19] and X-Compiler [20] enable students to promote their programming skills in simpler and less complicated environments than professional ones. Using graphical representations, many animation educational tools have been proposed, for instance SICAS [21], JAWAA [22], Jeliot 2000 [23], Raptor [24], and Choregraphe [25], to enable students to better understand programs. In addition, several tools have been developed to apply Artificial Intelligence (AI) techniques, such as DISCOVER [26] and Lisp Tutor [27] to support individualized learning. With these tools, students attain error and misconception findings as well as corrections in their programs through program simulation.

Even though it is believed that guiding students to identify and correct errors and misconceptions by simulating their own programs is valuable, students who are weaker cannot benefit from this because they are incapable of developing initial solution propositions to be simulated. It is worth mentioning that rather than focusing on problem solving skills, which are more essential for weaker students, such tools emphasize on programming language features more. In the hope of addressing the above issues, we developed the Flowchart-based Multi-agent System (FMAS) that benefit from an automatic text-to-flowchart conversion approach. It enables creating initial solutions for simple problems and improves problem solving skills. Some AI techniques are applied to FMAS development to convert the statement of a given programming problem (here in English) to a relevant flowchart. These techniques are: Natural Language Processing (NLP), Knowledge-based System, Knowledge Expansion, a Web Crawler (web monitoring service), and Multi-agent System. FMAS also contains a system chat and online chat (i.e. for a worst-case scenario), error system, instant feedback, synchronization of the text and its relevant sub-flowchart, and a visualization-based approach to support weaker students to create basic algorithms.

The rationale behind developing FMAS is to assist students in visualizing the relation between the problem statement (here in English text) and its pertinent flowchart while becoming engaged in the process of flowchart development which can be subsequently improved. Using FMAS, the students are guided through a dialogue system chat. FMAS encourages students by providing hints, synchronization, errors and extra information on the concept. This interaction urges step-by-step solution construction. As aforementioned, the target audience of this research involves CS minors who have no prior knowledge of programming. A novel flowchart-based multi-agent system that benefits from an automatic text-to-flowchart conversion approach to enhance CS minors' problem solving skills is the main contribution of this work. The proposed novel multi-agent system can be utilized in many academic applications, including problem solving, drawing diagrams, etc., as well as making the teaching of programming subjects more appealing for instructors.

The remaining parts of this paper are dedicated to the following sections: 2) mental model and visualization, explaining how flowcharts can provide novice programmers with clear mental models of algorithms; 3) FMAS architecture, elaborating main components and different scenarios of the proposed system; 4) evaluation, results and discussion which is dedicated to the devised experimental study, the results attained for both experimental and control group, and analyses of the results to see whether or not the use of FMAS in practice is supported and 5) conclusions and future work.

2.0 MENTAL MODEL AND VISUALIZATION

Mental models are very important in the comprehension of programming and other technical computing subjects. A student's success in preliminary programming courses is so much dependent and affected by developing the representations of process flow and mental models [28]. The importance of these models in developing understanding is also emphasized by Winslow [29]. In many cases, without having a mental model of the solution to a problem, students attempt to code the solution. In many cases, they are not provided with a cognitive or visual model in the learning context. In order to convey programming concepts, the majority of instructors emphasize on pseudo codes that offer somewhat good explanations for instruction in English. However, such pseudo code explanations cannot address program execution flow between the flowchart and program components. The lack of a mental model of execution can lead to considerable difficulties in understanding the relation among individual programming components. With the aim of visualizing the structure of programming, flowcharts have been used to overcome the inconvenience in translating a problem specification to its corresponding program code solution.

Flowcharts offer novice programmers clear mental models of algorithms without the need for prior training. As Westphal et al. [30] stated, "Even with having a pseudo code, it is so hard for novices to communicate the flow of a program unless using flowcharts or diagrams." For modeling large and complicated problems, flowcharts are not well-suited but are more appropriate for simple programs and conveying basic concepts to novice programmers. It can be said UML activity diagrams are more adequate modern visualizations than flowcharts, and are better for novice programmers to use. However, there is no privilege in terms of functionality or visualization for UML activity diagrams over flowcharts in the context of simple novice programs. Bassat Levy et al. [31] indicated that the effectiveness of flowcharts can be extended by using visualization-based tools and environments to assist novices with problem solving and program development. These environments provide novices with a concrete model of execution that is required by most to understand algorithms and programming. Preventing novice programmers from engaging in programming problem statements may lead to serious difficulties once faced with new programming exercises. More importantly, it is necessary for users to observe the relation between the textual forms of exercises given and relevant flowcharts. This provides users with the capability to relate a programming problem statement to a flowchart design more effectively, aiding with the transition from problem text specification to a relevant flowchart. A web-based environment that visualizes the solution development for a programming problem by converting the given problem statement to a relevant flowchart while engaging users in flowchart development, provides novices with an accurate mental model of execution. Consequently, an automatic text-to-flowchart conversion approach enables users to directly observe how a textual programming problem maps onto a flowchart.

3.0 THE ARCHITECTURE OF THE PROPOSED SYSTEM

Teaching computer programming to CS minors is supported by using FMAS, whose architecture is shown in Fig. 1. Highlighting the essential principles of various algorithms on a higher level of abstraction as well as problem solving ability through designing activities are the main aims of this system. FMAS has two different scenarios: a keyword is found and a keyword is NOT found. The former employs six agents along with five sub-agents to convert the programming problem statement into a corresponding flowchart while engaging novices in flowchart development. The latter uses four agents accompanied by two sub-agents in order to convert the text form of the programming problem to its relevant flowchart. In total, there are eight agents and seven sub-agents: the NLP, keyword finder, dictionary, flowchart, error detector, crawler, process orientation and admin agents; and synonym and substitution sub-agents, guidance and toolbar sub-agents, system chat sub-agent, and flowchart and online chat sub-agents. The role of each agent in the first scenario where a keyword is found is described and elaborated below.

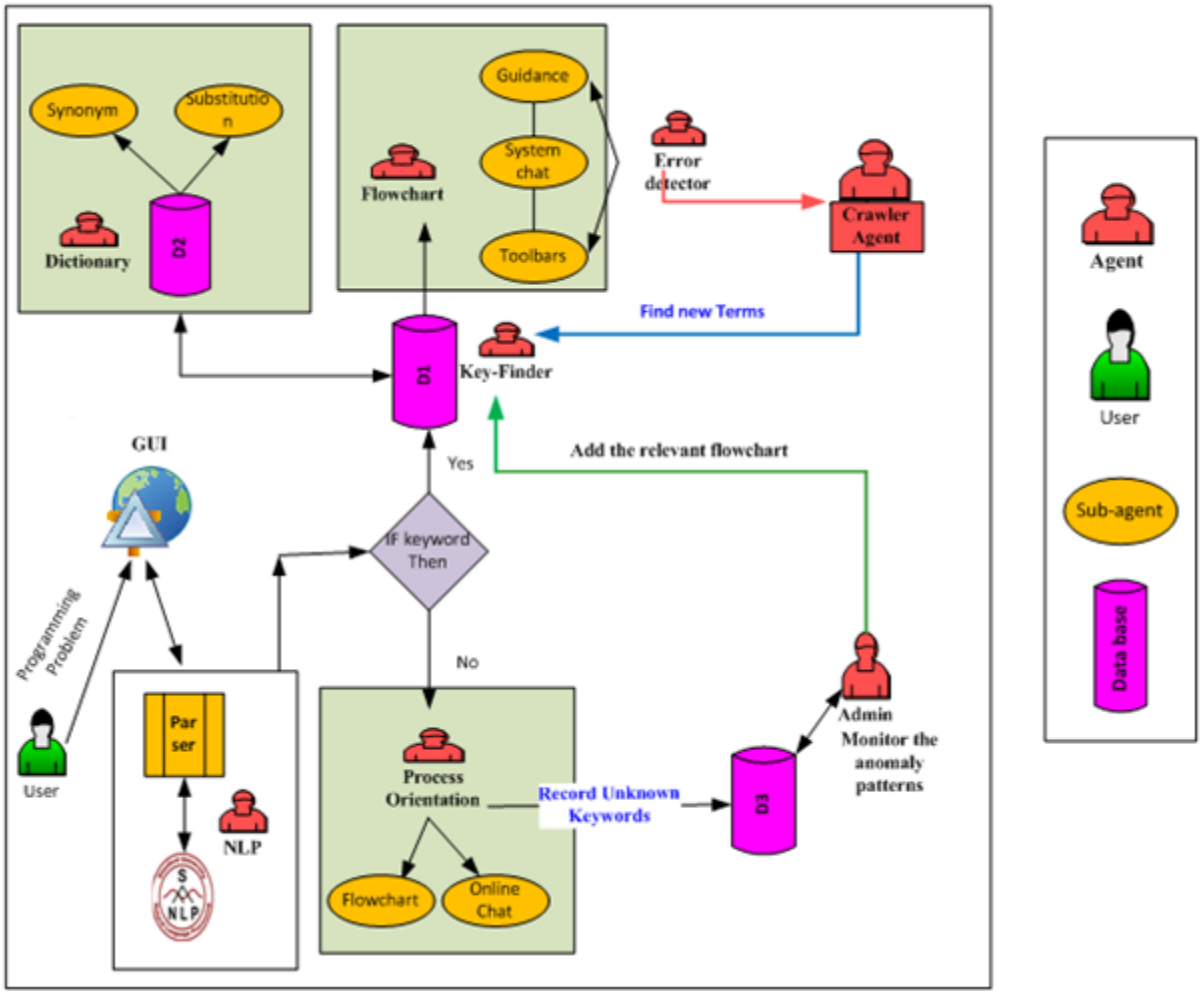

Fig 1. FMAS architecture

3.1 The First System Scenario

GUI (a software component) is the interface that a user will employ to interact with other agents. The flowchart and process orientation agents communicate with GUI. The content from the flowchart agent and process orientation agent includes the drawn flowchart or sub-flowchart along with instant feedback and messages will be passed to the user by GUI. This software component will take the problem given in text form and convey it to the next agent called the NLP agent.

3.1.1 NLP Agent

The NLP agent performs 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 carries out noise removal, including prepositions, conjunctions, etc., and only the main words will be handed to the key finder agent.

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

After parsing we have: *write/VB a/DT program/NN to/TO calculate/VB factorial/NN of/IN given/VBN number/NN*

As seen above, an online parser, Stanford Parser [32], processes the entered text, and the system automatically removes the noise in the parsed sentence. This means that prepositions, conjunctions, and so on, will be removed and only the main words will be chosen. Therefore, the output for example 1 after noise removal is:

Calculate/VB factorial/NN given/VBN number/NN

3.1.2 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 scenario starts working.

3.1.3 Dictionary Agent

This agent checks words for 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 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 find the largest among three numbers.*

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

- ***Substitution Sub-agent***

This sub-agent cross-checks the words with its repository and if it finds any substitution, it will pass it to the dictionary agent.

3.1.4 Flowchart Agent

This agent receives the keyword found from D1 and uses a draw-able representation module to provide the GUI software component with a workspace for users to complete the sub-flowchart, a system chat for step-by-step guidance, and the flowchart template. System assessment provides both a means to guide student learning and feedback for the learner about the learning process. The flowchart agent includes three sub-agents as follows:

- ***Guidance Sub-agent***

This sub-agent offers users a workspace, a flowchart template with sub-flowcharts located in the right places, various flowchart notations for dragging and dropping to complete the flowchart template, instant feedback in the system chat, extracting the correct text and content of each shape in the flowchart from D1 and placing it in the shape dropped by users, and showing the full flowchart to be compared and traced by users. The workspace generated for example 1, different shapes to complete the flowchart, instant system error and feedback along with the system chat are shown in Fig 2. It should be noted that this sub-agent does not allow the wrong shape or notation to be dropped in the flowchart template and gives users an instant error and feedback. Upon dropping the correct shape in the template, suitable and relevant content or text will be extracted from D1 and placed in the shape dropped to facilitate full user guidance.

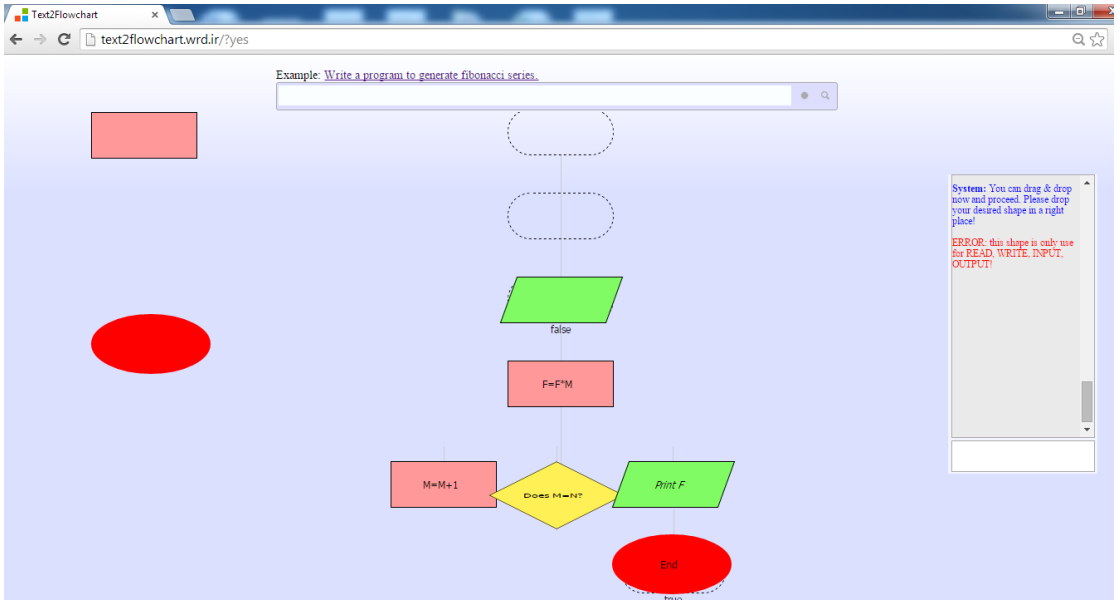

Fig 2. Workspace provided by the guidance sub-agent

Once users complete the flowchart, the system asks whether they wish to have a full flowchart extracted from the Internet [33], as shown in Fig 3. This option enables viewing the correct flowchart next to that developed by the users, who can then compare and trace flowcharts. In addition, correct arrows are visible in the flowchart from the Internet.

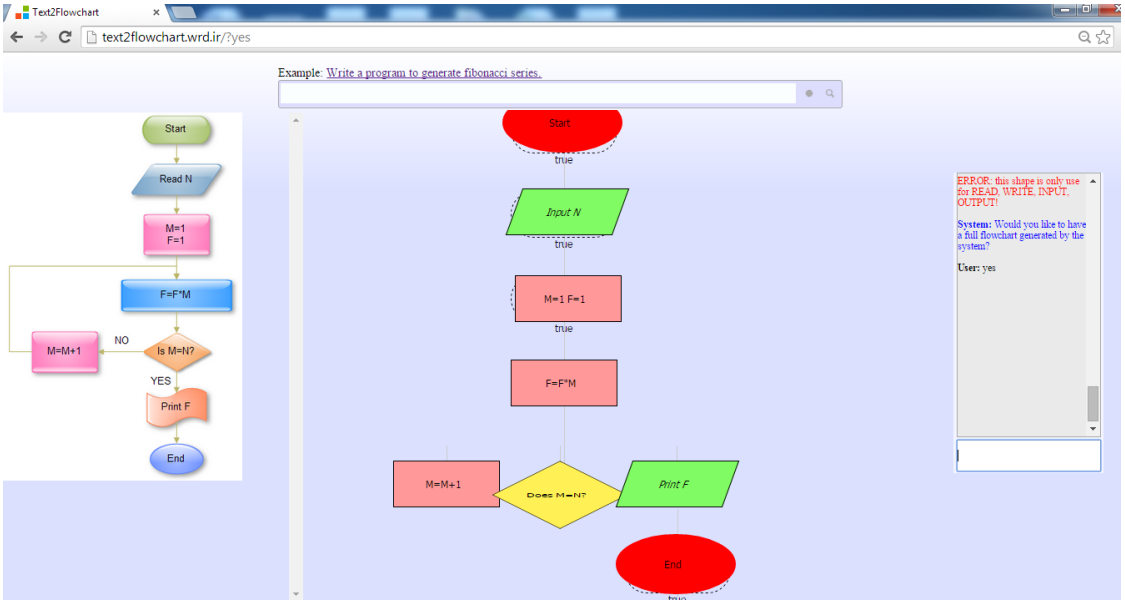

Fig 3. Workspace provided by the guidance sub-agent along with full flowchart from the Internet

- ***Toolbar Sub-agent***

This sub-agent provides users with the workspace, flowchart template, and various flowchart notations for dragging and dropping to complete the flowchart template, and brief feedback next to each shape after flowchart completion. The workspace generated for example 1, different shapes to complete the flowchart, system error upon flowchart completion along with the system chat are illustrated in Fig 4. This sub-agent only provides the workspace and flowchart shapes for users, who will receive brief feedback on their task only after completing the drag and drop and

not instantly. In addition, users can enter text through each shape dropped in the flowchart template, meaning that the system does not fully guide them.

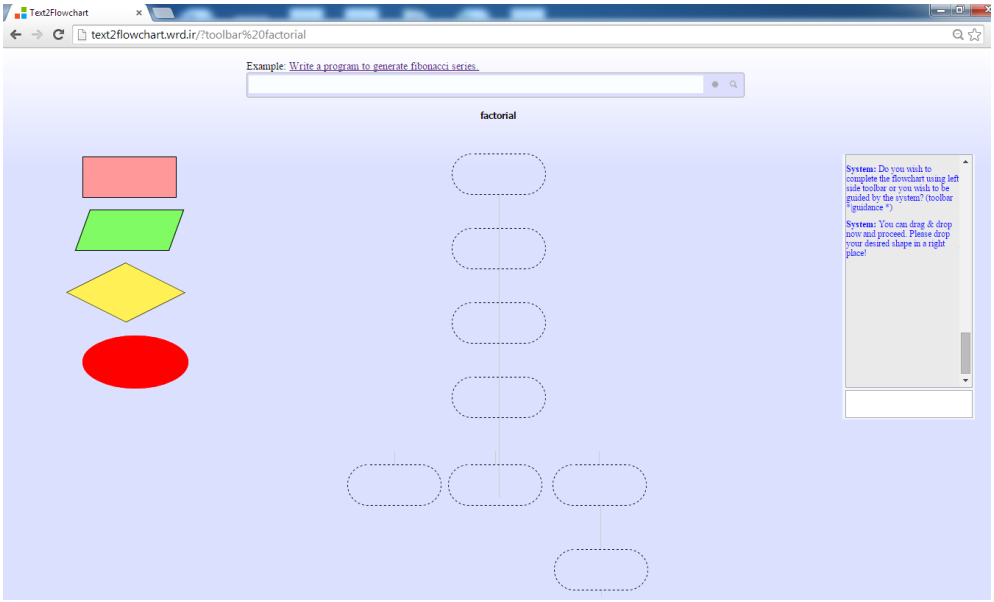

Fig 4. Workspace provided by the toolbar sub-agent

Feedback upon flowchart completion is presented in Fig 5 along with an automatically generated question by the system chat which asks whether the user wishes step-by-step guidance from the system through the guidance sub-agent. Another classical way of making users think after a failure is to limit the amount of feedback.

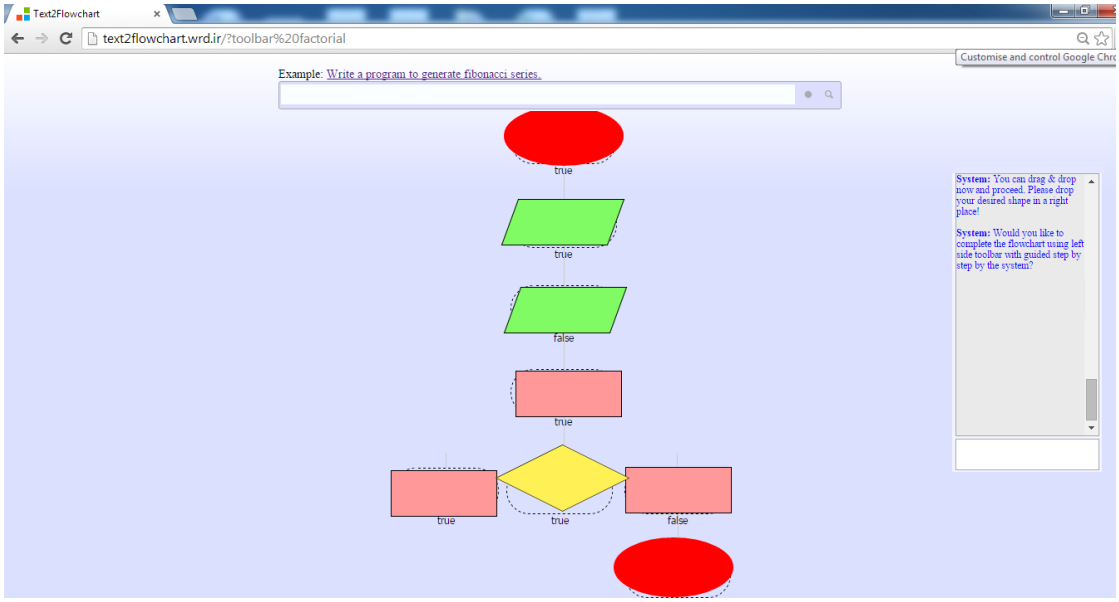

Fig 5. Toolbar sub-agent workspace with brief feedback

- ***System Chat Agent***

This agent provides users with immediate feedback, errors and recommendations while they are completing the flowchart (Figs 2 and 3).

3.1.5 Error Detection Agent

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

3.1.6 Crawler Agent

The crawler agent receives an unrecognized keyword from the error detection keyword, and crawls relevant websites (e.g. [33]) to find a related definition and context for improving the D1 database. Once this agent extracts additional information, it will be automatically added 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. Fig 6 illustrates the function of these two agents in example 1.

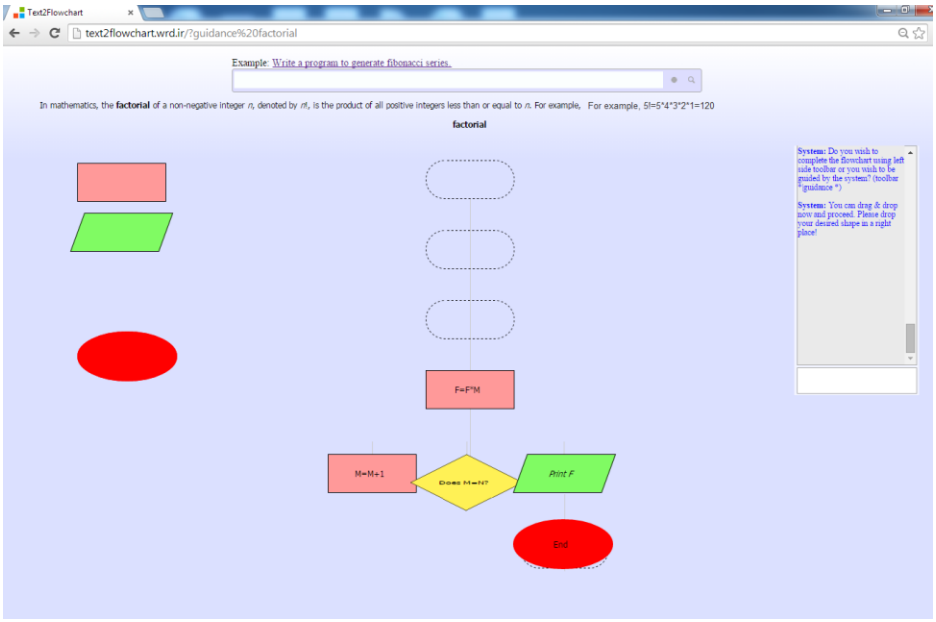

Fig 6. Workspace of the guidance sub-agent with extra information added by the crawler agent

3.2 The Second System Scenario

The role of each agent in the second scenario for 'a keyword is NOT found,' is presented.

3.2.1 NLP Agent

The GUI software component and NLP agent are the same as described in Section 3.1.
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 a/DT program/NN that/WDT asks/VBZ the/DT user/NN to/TO type/VB an/DT integer/NN and/CC write/VB _/VBG you/PRP win/VB _/NNS if/IN the/DT value/NN is/VBZ between/IN 56/CD and/CC 78/CD*

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 scenario of FMAS differs from the first scenario. Therefore, the output for example 3 after noise removal is:

Asks/VBZ user/NN type/VB integer/NN write/VB _/VBG you/PRP win/VB _/NNS if/IN value/NN is/VBZ between/IN 56/CD 78/CD

3.2.2 Key Finder Agent

This agent cross-checks the main words extracted from the sentence entered by the NLP agent with keywords stored in D1and D2. If no match is detected, the main words extracted from the programming problem statement will be sent to the process orientation agent.

3.2.3 Process Orientation Agent

In case no keyword is found, this agent obtains the main words from the NLP agent, refers each related word to its corresponding flowchart notation (for example, if there is '?' or "If" in the problem statement, a diamond will be drawn, in which the flowchart sub-agent will place relevant words) 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 software component with a workspace for users to complete the sub-flowchart, an online system chat to guide users step-wise and a flowchart template. For example 3, the process orientation agent automatically generates relevant sub-flowcharts using the flowchart sub-agent presented in Fig 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.

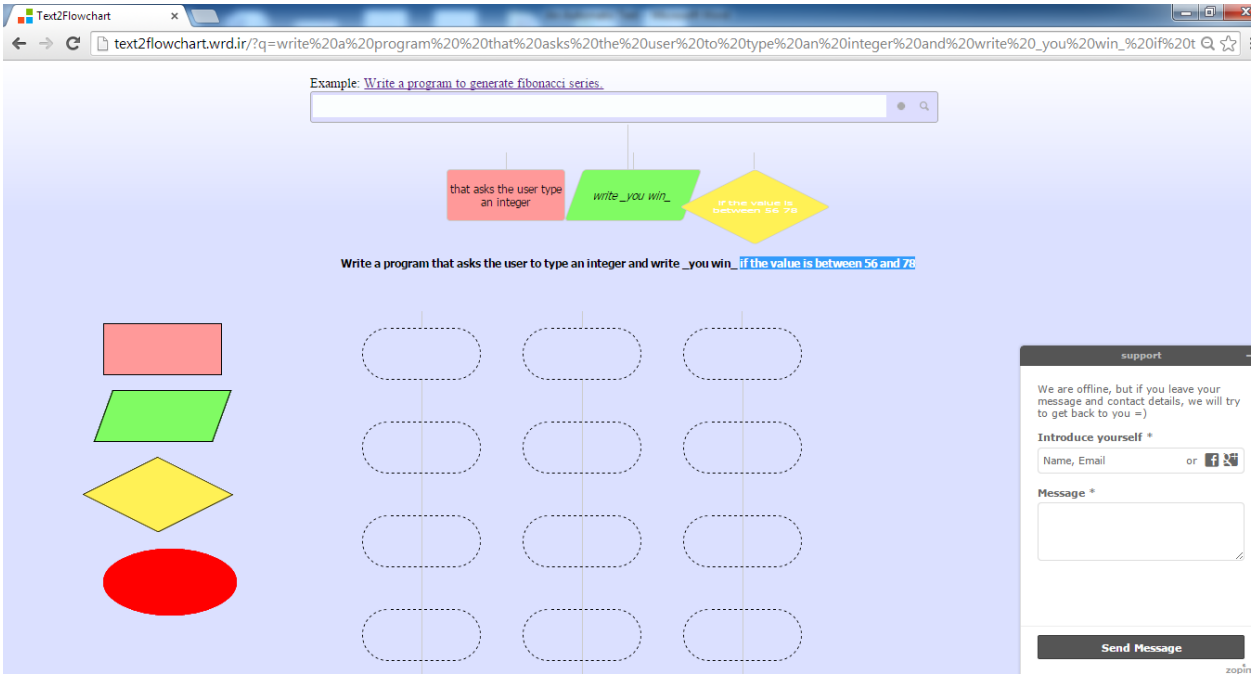

Fig 7. Workspace of the process orientation agent

- ***Online Chat Sub-agent***

It is not unusual for students to get stuck on certain programming assignment stages. In such situations, it is important to get timely help from an instructor to be able to continue working on the assignment. Otherwise, students may give up or not have sufficient time remaining. Therefore, an online chat sub-agent provides novices with an online chat with the system's admin for further flowchart development. As shown in Fig. 7, an online chat is improvised at the bottom right side of the page to help users obtain more guidance from the system admin. If databases 1 and 2 are input properly and have 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 worst-case scenarios.

3.2.4 Admin Agent

The admin agent requests 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.

4.0 EVALUATION AND PARTICIPANTS

In total, 50 first-year undergraduate students from the University of Malaya, with no prior knowledge of computer programming, participated in the current study. The study duration comprised two, two-hour sessions every week conducted over two weeks. To improve the problem solving skills of participants while reducing the syntactical burden inherent to programming languages was the initial hypothesis. After a traditional instruction presentation in the first session, the participants were supposed to solve three programming problems. Afterwards, the participants were introduced to FMAS with a 30 minute lecture in the second session and were then asked to use the system to solve the desired exercises. In the third and fourth sessions, they were given simple and basic programming problems with increasing complexity. The participants were asked to solve 2 simple programming problems in order to get introduced to using FMAS and a few programming concepts. Examples of special cases were provided for clarification. Evaluation was conducted in an informal setting by ten evaluators every two hours. Different evaluation approaches such as problem solving monitoring, questionnaires, observation, and interviews, were used to evaluate the proposed system.

4.1 Instruments and Data Collection

The learning materials presented to students consist of some theoretical knowledge in basic computer programming, algorithm and solution design, and flowchart development. These learning materials were taught during two separate lecture and practical sessions. The participants were divided into two groups. Two key factors considered in FMAS evaluation were usability and effectiveness. The usability of FMAS was regarded as similar to any other software applications. Jacob Nielsen [34] indicated that software application usability is employed to assess how easy it is to use user interfaces. Thus, we categorized the feedback gained from the evaluation into five areas: ease of use, error handling, enjoyment, reliability, and website-related questions. With regard to FMAS effectiveness and whether it is educationally beneficial to novice programmers, we evaluated the efficacy of FMAS, which is the most important factor. In this study, three subjective data gathering techniques were applied, namely questionnaires, observation, and interviews, besides an objective data gathering technique, i.e. problem solving monitoring, to determine the effect of FMAS on improving novice programmers' problem solving skills. As for the objective data gathering technique, the two methods applied were monitoring the task completion time and number of solved problems in each session [35, 36].

5.0 RESULTS AND DISCUSSION

This research aims to generate feedback regarding FMAS usability, efficacy, and problem solving ability. The questionnaire data gathered from the study, mean, standard deviation, variance as well as response description and frequency are presented in Table 1 and Fig 8. In order to combine the results of the questionnaires using different scales, the data is presented in percentage. The questionnaire was divided into usability and efficacy groups.

Table 1: Questionnaire Results

No	Study Aim	Evaluation Criteria	Mean	Std. Deviation	Variance
1	**Usability**	This programming tool is enjoyable to use	3.9800	.55291	.306
2	**Usability**	This programming tool is easy to use and understand (I can learn how to use the tool within 5 minutes)	4.3600	.85141	.725
3	**Usability**	The tool performs its function in a correct and efficient manner	4.3800	.63535	.404
4	**Usability**	The tool is speedy and responsive	4.1000	.50508	.255
5	**Usability**	The use of color is beneficial (The design of this tool is attractive)	4.1400	.72871	.531
6	**Usability**	Launching the tool is easy	4.2800	.64015	.410
7	**Usability**	The user interface design is appropriate for an inexperienced user	4.1200	.84853	.720
8	**Usability**	The animation helped me develop a solution and understand how a program works	4.3600	.74942	.562
9	**Usability**	I enjoyed solving the programming problem using this tool	4.6400	.48487	.235
10	**Usability**	System chat and instant feedback are helpful	4.4400	.70450	.496
11	**Usability**	The error messages are helpful to provide guidance to correct my mistake	4.1400	.63920	.409
12	**Usability**	The programming problems given are at the right difficulty level for me	4.0200	.86873	.755
13	**Efficacy**	The automatic text-to-flowchart approach is helpful when I have no idea about the solution	4.2600	.98582	.972
14	**Efficacy**	The flowchart visualization helped me when developing a solution	4.5400	.50346	.253
15	**Efficacy**	I had few problems learning how to use the tool to develop my solution	3.0200	.79514	.632
16	**Efficacy**	The flowchart is useful for designing computer programs and sharing ideas	4.6000	.49487	.245
17	**Efficacy**	I understand the relationship between the flowchart and programming problem statement after using the tool	4.5400	.64555	.417
18	**Efficacy**	The tool made the programming concept easier to understand	4.2600	.69429	.482
19	**Efficacy**	I would recommend the tool to others who want to learn programming	4.5400	.50346	.253
20	**Efficacy**	The tool enabled me to see a design solution	4.4400	.73290	.537
21	**Efficacy**	The flowchart enabled me to understand the solution being developed	4.6000	.49487	.245

22	**Efficacy**	I feel I have learnt some skills by using the tool and solving the problems	4.1600	.68094	.464
23	**Efficacy**	The tool has positively influenced my interest in programming	3.9800	.95810	.918
24	**Efficacy**	The tool helped me understand how to design a solution in programming	4.3000	.61445	.378
25	**Efficacy**	The tool helped me develop and improve my solution designing and problem-solving skills	4.3000	.76265	.582
26	**Efficacy**	The tool increased my understanding of computer programming	4.2800	.57286	.328
27	**Efficacy**	The tool as a whole and its features helped me overcome conceptual difficulties in programming	4.3600	.69282	.480
28	**Efficacy**	The tool enables me to focus and improve my problem solving skills	4.4400	.64397	.415

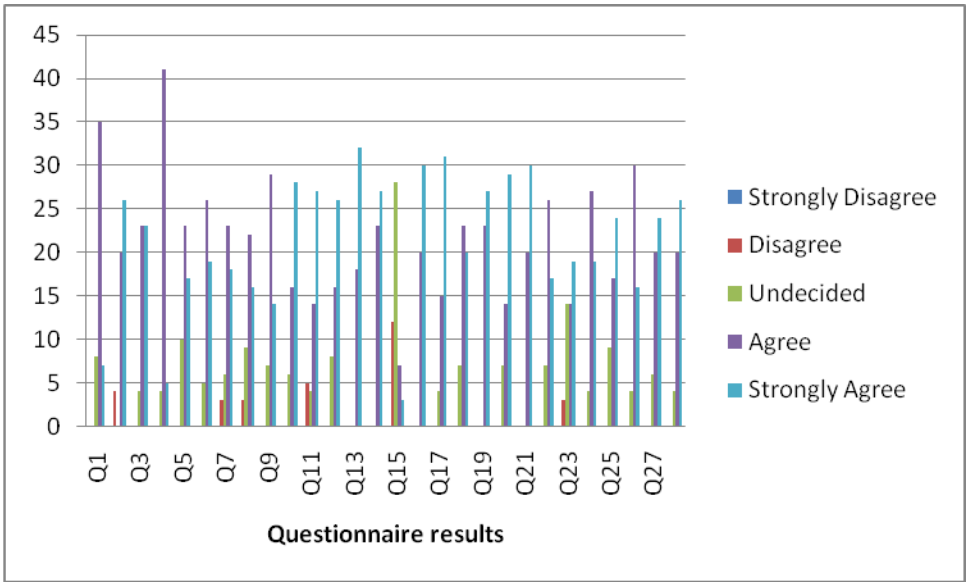

Fig 8. Questionnaire data gathered from 50 participants

The reliability of the result should be studied upon collecting the evaluation results. Various methods have so far been applied to measure reliability, and **Cronbach's Alpha** is used in this research. This method is normally used to measure internal consistency and the range is between 0 and 1. As long as α is close to 1, it is said to be reliable [37] but overall, this range should be higher than 0.7 to prove reliability. The Cronbach's alpha measure for the usability and efficacy of the questions shown in Table 2 is greater than 0.7, which indicates the high reliability level of the questionnaire and implies sufficient internal consistencies have been judged for a reliable measure.

Table 2: Reliability Statistics

Factor	Cronbach's Alpha
Usability	.947
Efficacy	.972

In general, the data collected indicates the research positivity. The main results from the samples as a whole and individually for each factor are shown in Fig. 9, where bars represent the average score assigned to each item.

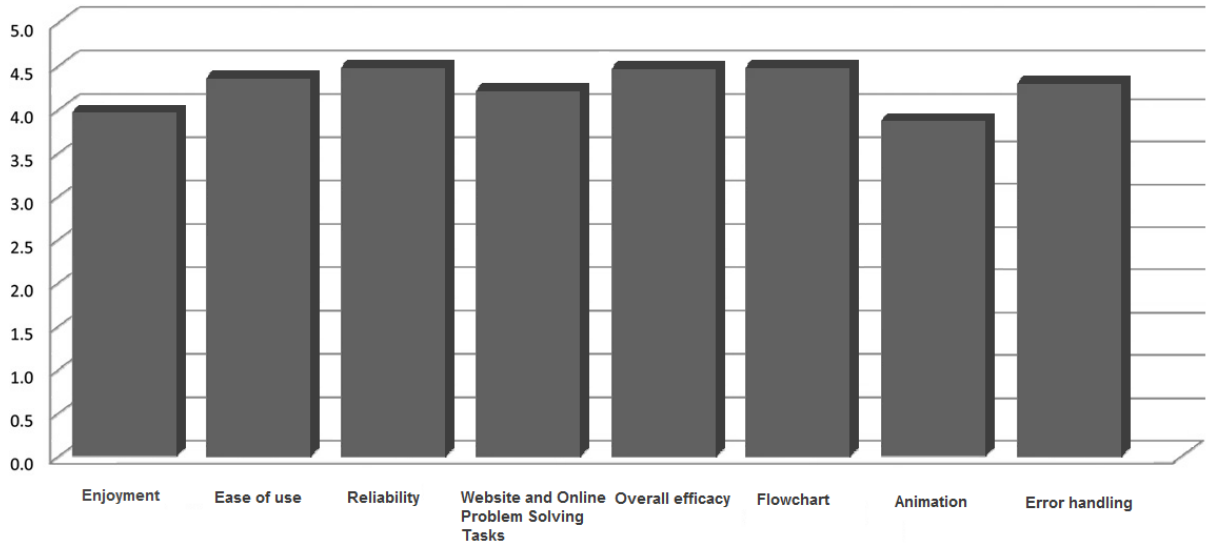

Fig 9. Results of students' opinion about FMAS

The diagram above signifies that participants liked learning programming using FMAS and it helped them enjoy problem solving and solution designing activities with ease of use. Other factors, such as web and online problem solving tasks, overall efficacy, flowchart, animation, and error handling were deemed highly positive by users. Most students reported high satisfaction with FMAS. Table 3 provides additional details on the mean and standard deviation concerning various factors. Regarding the efficacy of FMAS in improving problem-solving skills, which is considered the main goal of this research, the results indicate that the overall system efficacy in problem solving and solution designing activities is 4.6000. This signifies the success of FMAS in attaining the main aim of the research.

Table 3: Mean (M) and Standard deviation (SD) for items about students' opinion of FMAS

Items	Usability	Efficacy	M	SD
Enjoyment (Questions 1, 9)	*		3.9800	0.782
Ease of use (Questions 2, 5, 7)	*		4.3600	0.765
Reliability (Questions 3, 4, 12)	*		4.6400	0.870
Website and Online Problem Solving Tasks (Question 6)	*		4.0200	0.910
Overall efficacy (Questions 13, 14, 18, 19, 20, 22-28)		*	4.6000	1.028
Flowchart (Questions 16, 21)		*	4.6400	1.037
Animation (Questions 8, 15)		*	3.7600	0.980
Error handling (Questions 10, 11)	*		4.3000	1.030

5.1 Analysis of Usability

The enjoyment and simplicity of FMAS were liked by participants and are considered among the most significant evaluation aspects. Regarding system enjoyment, the participants were evaluated according to responses to Questions 1 and 9. Question 9 emphasized problem solving enjoyment using FMAS and the majority of participants considered it one of the system's strengths. Question 1 merely focused on the enjoyment factor and the high response rate to this question indicates that participants found FMAS enjoyable and interesting. Therefore, a positive overall result was obtained from these questions. Questions 2, 5, 7 and 15 assessed the ease of use and simplicity of the system. In the evaluation conducted, the participants' responses showed the highest study average, indicating that FMAS is easy to use. Observations and interviews with the evaluators regarding these factors show that they liked the simplicity of FMAS; however, some stated that the interface should be improved in certain sections. For example, three evaluators asked for save and undo options. Participants' responses to questions 3 and 4 showed the system's reliability. The feedback received to these questions was satisfactory. Nonetheless, perfect software of 100% with Likert evaluation is nearly impossible. Some minor problems were found and mitigated during FMAS development. Thus, the score is deemed a good indicator of reliability with minor room for improvement. Views on website and online problem solving tasks were obtained from question 6. The high response rate shows a good performance level in this area. The participants' responses to question 11 were also positive, as expected, meaning the system is strong enough to handle errors.

5.2 Analysis of Efficacy

Questions 10, 12, 18-20, and 22-28 directly addressed the efficacy of FMAS as a teaching aid. Overall, the feedback gained regarding system efficacy illustrates that FMAS is regarded as useful and helpful in particular. System chat and instant feedback generated by the system as well as animation were evaluated using questions 8 and 10. Surprisingly, the responses to these questions had some of the highest rates, signifying that color is considered a significant aid to participants' understanding. Feedback from the interview concerning this question indicates that differentiation between various flowchart components was the main reason for this finding. Additionally, applying various colors eases spotting the key functionality of an algorithm modeled in FMAS. The rest of the questions regarding system efficacy were aimed at exploring the system's effectiveness and efficacy in problem solving and solution designing activities. Questions 13 and 17 showed the importance of the novel, proposed text-to-flowchart

approach, and their results showed that FMAS considerably encourages and enables students to engage in problem solving activities. During observation and interviews, the participants were asked about the new approach applied in the system. They gave positive feedback, indicating the effectiveness of this approach, specifically while students are off-campus and are required to do exercises when they have no idea regarding the solutions. They all declared that having guidance through solution development by means of flowchart drawing is the best for CS minors. FMAS shows clear relations between the programming problem statement and its relevant components in a flowchart by synchronized highlights and side-by-side views. For this feature, question 17 was designed for the participants. Feedback gained from the questionnaire and observations shows that after using FMAS, the participants felt they had some sort of understanding of the solution being developed. As a result, this feature was emphasized more prominently in subsequent evaluations and resulted in a much stronger response. The power of the flowchart as a teaching aid for participants is displayed through questions 14, 16, and 21, where the usefulness of flowcharts in FMAS was examined. More than 90% positive responses demonstrated the effectiveness and suitability of flowcharts for CS minors.

5.3 Analysis of Problems Solving

In this section, problem solving activities along with their efficacy are assessed. As indicated in the previous sections, in order to monitor participants' problem solving activities, the improvement in the 10 pre-selected participants' completion times (Table 4) and the number of problems solved in each session by 32 pre-selected participants (Fig 10) were monitored and assessed. Ten participants were pre-selected based on their level of knowledge for monitoring the improvement in completion time; while 32 participants were pre-selected based on their attendance in each session for monitoring the numbers of problems solved in each session.

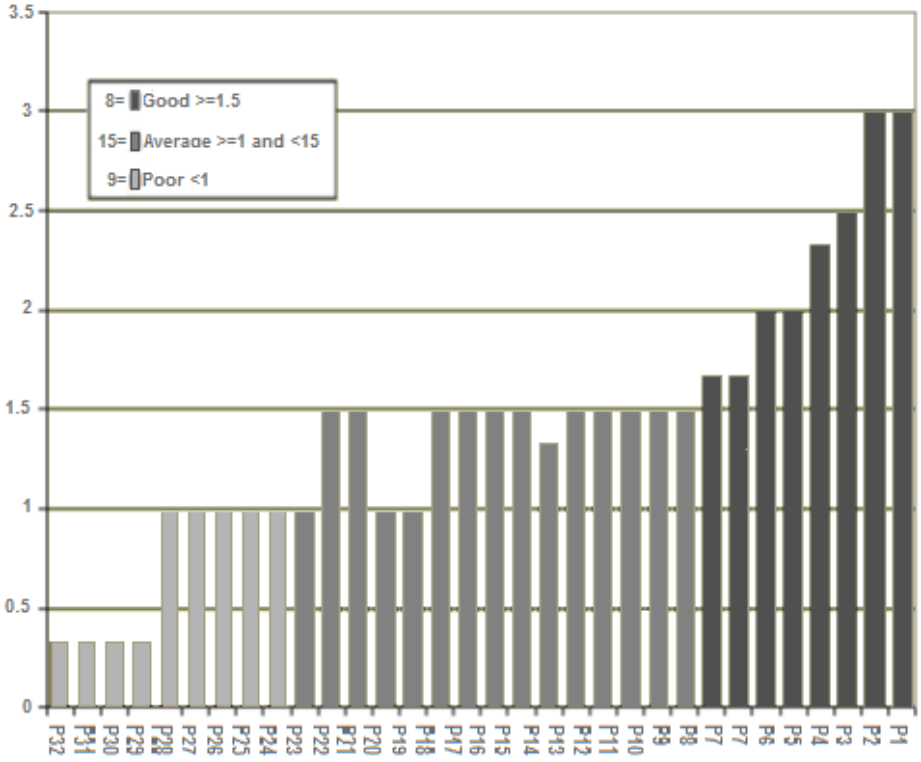

Fig 10. Average number of problems completed per session

Once learning difficulties were tackled, completion time decreased although problem complexity increased. Regarding the number of problems solved per session, by using the system, the majority of participants were at least

able to solve one or two of the given programming problems. This shows the effectiveness of FMAS in problem solving activities. A few participants were unable to solve problems before using FMAS, but afterwards, their problem solving abilities enhanced and they solved at least one problem per session. In general, the feedback and results collected from problem solving monitoring indicate that the participants found FMAS very effective and enjoyable. It should also be mentioned that some participants responded with a low positive. The time limitation caused some of the participants to spend less time familiarizing with FMAS. Other factors affecting the evaluation results were age and gender, which should be investigated further.

Despite the fact that the problems grew in complexity from one to the next (Problem 1: Max and Min; Problem 2: Factorial; Problem 3: Area and Perimeter of Square), the time completion rate results shown in Table 4 demonstrate a decline in completion time with each successive problem. During problem solving monitoring, four participants were not able to complete problems a2 and a3. However, by using FMAS, not only could they tackle this issue, the time decrement in completion time for problems b2 and b3 was a real surprise. This shows that FMAS and the problem solving activities had a very positive impact on most participants' problem solving skills.

Table 4: Time completion rates of 10 participants in two different sessions

Participant No	Problem a1	Problem a2	Problem a3	Problem b1	Problem b2	Problem b3
1	2:45 min	2:20 min	3:00 min	2:40 min	2:10 min	3:00 min
2	3:10 min	3:00 min		1:55 min	2:10 min	2:45 min
3	1:50 min	2:30 min	2:40 min	1:55 min	2:50 min	2:30 min
4	2:00 min	2:20 min	2:50 min	2:00 min	1:50 min	2:00 min
5	4:00 min			2:30 min	2:40 min	2:30 min
6	1:30 min	2:40 min	3:00 min	1:10 min	1:30 min	1:40 min
7	2:10 min	1:50 min	2:00 min	2:00 min	2:10 min	2:10 min
8	2:40 min	2:50 min	2:30 min	1:40 min	1:55 min	2:15 min
9	2:00 min	3:10 min		1:20 min	1:55 min	1:55 min
10	2:30 min	2:00 min	2:30 min	2:00 min	2:00 min	2:20 min
Max	4:00 min	3:10 min	3:00 min	2:40 min	2:50 min	3:00 min
Min	1:50 min	1:50 min	2:00 min	1:10 min	1:30 min	1:40 min
Avg. Time	2:27 min	2:32 min	2:38 min	1:55 min	2:07 min	2:18 min
No	10	9	7	10	10	10

According to the results of various evaluation techniques conducted, FMAS, which benefits from an automatic text-to-flowchart approach, is a motivating and effective way of teaching introductory programming to novice programmers. Although a tool may be considered effective, it will not be adopted as long as the overhead of doing so is high. This study focused on discovering if FMAS could yield some improvements to the problem solving skills of novice programmers in the basics of imperative programming. Furthermore, more than 90% of participants agreed that during the interview and questionnaire, FMAS would make it easier for inexperienced users to learn

programming. A particularly interesting finding was that FMAS appeared to be suitable for nearly all participants. By examining the evidence collected from the evaluation, it became clear that FMAS along with its features and strategy of user involvement in flowchart development are very well-suited for the CS minors participating in this study. It is also clear that by using FMAS and solving the exercises, the students became more interested in programming concepts whilst developing problem solving skills. This is evidenced by their decreasing completion time and feedback to the questionnaire. The study provided further evidence of the FMAS efficacy. From the questionnaire, observations, and interviews, it is apparent that the evaluators also gained a general understanding of solution development. This knowledge was acquired primarily through the evaluators' active engagement with FMAS via problem solving tasks, demonstrating the efficacy of FMAS in problem solving. This was due to the intuitiveness of the flowcharts, as the majority of evaluators agreed that they were enabled to understand the solutions being developed. Rather surprisingly, many of the participants considered the flowchart use of color beneficial. The efficacy of the web launching features was also evident in this study. The developed website was additionally seen as effective and very appropriate for hosting online activity packs and problems. A number of participants faced some difficulties in expression entry; however, this subsided after completing the second problem. Such errors are more typographical than logical. In summing up the error handling features, while many users found FMAS's error messages helpful, some mentioned that it could be further simplified. The efficacy of the flowcharts became apparent in the study findings. Feedback from the students indicated that the animation features were useful, insightful and helpful to developing program solutions. The synchronization of the programming problem statement and its relevant sub-flowchart was easily visible to users, which enabled them to understand the real connection between the flowchart and texture of the programming problem as well as structure semantics. Despite the study having been conducted outside the campus network, the results show that the web launching features of FMAS to be very effective, as 20 students accessed the tool simultaneously in approximately one minute. The efficacy results indicate that the participants strongly believed they were learning something from the whole experience. They also strongly believed that FMAS is a good tool for learning programming, and they would recommend it to friends and would like to use it in their school. According to the results, the automatic text-to-flowchart conversion approach is viewed as a very useful aid to comprehending and problem solving, especially by those performing well in problem solving tasks. Similarly, animation was deemed as an aid to understanding but also a very motivating and rewarding activity. Regarding problem solving improvement in both completion time and the number of problems solved, the participants of this study provided very positive feedback on average. Nearly all participants faced one or two learning difficulties and after using FMAS not only did the number of problems solved per session increase, but the time completion rate declined steadily as well. It is evident that the participants' problem solving skills enhanced by engaging with FMAS. Still, a lack of adequate introduction to FMAS led to some confusion for the participants, which was resolved with further explanation. To sum up, the enjoyment levels experienced by the participants coupled with improvements in their problem solving ability demonstrates that FMAS is an effective and motivating tool for introducing programming.

6.0 CONCLUSION AND FUTURE WORK

A number of students encounter various difficulties in the preliminary learning stages and are unable to develop solutions for simple programming problems. This might result in giving up and losing interest, which can lead to dropping out and higher failure rates. FMAS is a flowchart-based multi-agent system meant to support problem-solving skills in the form of flowchart development for basic and simple programming problems aimed at CS minors. With FMAS, students are involved in developing flowcharts using an automatic text-to-flowchart conversion approach that contributes to improving their problem solving skills. The system provides stepwise guidance, offering additional information regarding the entered programming problem and obtaining feedback for actions. The proposed approach applied in FMAS is an advance and improvement over many existing visual programming environments. An E-learning environment that visualizes the solution construction for a programming problem by automatically converting the given problem statement to its relevant flowchart while engaging users in flowchart development, will provide novices with an accurate mental model of execution. Thus, the main aim of this study was to support the problem solving ability through designing activities. We believe that FMAS is very successful because the criteria were designed carefully. The system benefits from a mature repository of basic and

fundamental programming problems aimed at CS minors along with a novel approach of automatic text-to-flowchart conversion, which enables FMAS to get novices involved in flowchart development. There are not many developed visualization tools intended for students with no prior knowledge of programming. Besides our system, a few others have been developed, such as SICAS [21] and RAPTOR [24]. They are similar to FMAS in a sense that they provide students with an environment for flowchart construction and visualization. However, the novel approach applied in FMAS distinguishes it from other related works. CS minors who do not know anything about programming are sometimes unable to use the aforementioned tools as they require some sort of user knowledge regarding the entered programming problems. FMAS resolves this inconvenience using a web-based environment to get users involved in flowchart development of the entered programming problem by offering three options. Even worst-case scenarios are improvised in FMAS in order to fully assist users, also in terms of problems that are not stored in the main system repository. Additionally, FMAS automatically improves its repository using an extra database to store the unknown entered programming problems along with web crawlers to enhance its main database. Finally, an experimental study was devised to assess the success of FMAS, showing very positive feedback. Therefore, the use of FMAS in practice is supported, as the results indicate considerable gains for the experimental group over the control group. A very awarding finding was that an automatic text-to-flowchart conversion approach applied in FMAS successfully motivated almost all participants in problem solving activities. Consequently, the results suggest further development of our proposed approach in the form of an Intelligent Tutoring System (ITS) in future to make the early stages of learning programming more encouraging for students.

ACKNOWLEDGMENTS

This work has been financially funded by the University of Malaya with project number of RG327-15AFR.

REFERENCES

[1] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz, , “A multinational, multi-institutional study of assessment of programming skills of first-year CS students”, in Proc. of 6th Annu. Conf. on Innovation and Technology in Computer Science Education, 2001, pp: 125-180.

[2] Soloway, Elliot, and James C. Spohrer, eds. Studying the novice programmer. Psychology Press, 2013.

[3] J. Carter and T. Jenkins, “Gender and programming: What's going on?”, in Proc. of 4th Annu. Conf. on Innovation and Technology in Computer Science Education, 1999, pp: 1-4.

[4] R. Moser, “A fantasy adventure game as a learning environment: why learning to program is so difficult and what can be done about it”, ACM SIGCSE Bulletin, vol. 29, 1997, pp: 114-116.

[5] N. Pillay, “Developing intelligent programming tutors for novice programmers”, ACM SIGCSE Bulletin, vol. 35, 2003, pp: 78-82.

[6] N. Pillay and V. Jugoo, “An Investigation into Student characteristics Affecting Novice Programming Performance”, ACM SIGCSE Bulletin, vol. 37, 2005, pp:107-110.

[7] Hooshyar, D., Ahmad, R. B., Shamshirband, S., Yousefi, M., & Horng, S. J. A flowchart-based programming environment for improving problem solving skills of Cs minors in computer programming. The Asian International Journal of Life Sciences, vol. 24(2), 2015, pp: 629–646.

[8] Dillon, Edward, Monica Anderson, and Marcus Brown. “Comparing mental models of novice programmers when using visual and command line environments.” Proceedings of the 50th Annual Southeast Regional Conference. ACM, 2012.

[9] T. Collett, "Augmented reality visualisation for player," Ph.D. thesis, Department of Computer and Elecrical Engineering, University of Auckland, Auckland, NZ, 2007.

[10] I. Boada, J. Soler, F. Prados, and J. Poch, "A teaching/learning support tool for introductory programming courses", in Proc. of 5th Int. Conf. on Information Technology Based Higher Education and Training, 2004, pp: 604-609.

[11] M. Ben-Ari, "Constructivism in computer science education", Journal of Computers in Mathematics & Science Teaching, vol. 20, 2001, pp: 45-73.

[12] Hooshyar, D., T. Maíen and M. Masih. Flowchart-based programming environments aimed at novices. International Journal of Innovative Ideas, vol. 13(1), 2013, pp: 52-62.

[13] Hooshyar, D., Ahmad, R. B., & Nasir, M. H. N. M. A Framework for Automatic Text-to-Flowchart Conversion: A Novel Teaching Aid for Novice Programmers. International Conference on Computer, Control, Informatics and Its Applications (IC3INA), Bandung, Indonesia 21–23 October 2014. IEEE, pp: 7–12.

[14] S. Cooper, W. Dann, and R. Pausch, "Alice: a 3-D tool for introductory programming concepts", Journal of Computing in Small Colleges, vol. 15, 2000, pp: 107-116.

[15] C. Kelleher, D. Cosgrove, D. Culyba, C. Forlines, J. Pratt, and R. Pausch, "Alice2: Programming without Syntax Errors", User Interface Software and Technology, 2002.

[16] D. Buck and D. J. Stucki, "JKarelRobot: a case study in supporting levels of cognitive development in the computer science curriculum", ACM SIGCSE Bulletin, vol. 33, 2001, pp: 16-20.

[17] Diprose, James P., Bruce A. MacDonald, and John G. Hosking. "Ruru: A spatial and interactive visual programming language for novice robot programming." Visual Languages and Human-Centric Computing (VL/HCC), IEEE Symposium on. IEEE, 2011.

[18] M. Kölling, B. Quig, A. Patterson, and J. Rosenberg, "The BlueJ system and its pedagogy", Journal of Computer Science Education, vol. 12, 2003, pp: 249-268.

[19] K. Van Haaster and D. Hagan, "Teaching and learning with BlueJ: an Evaluation of a Pedagogical Tool", in Proc. of the Information Science and Information Technology Education Joint Conf., 2004, pp : 455- 470.

[20] G. Evangelidis, V. Dagdilelis, M. Satratzemi, and V. Efopoulos, "XCompiler: Yet Another Integrated Novice Programming Environment", in Proc. of 2nd IEEE Int. Conf. on Advanced Learning Technologies, 2001, pp: 166-169.

[21] A. Gomes and A. J. Mendes, "Suporte à aprendizagem da programação com o ambiente SICAS", in Proc. of V Congresso Ibero-Americano de Informática Educativa, 2000.

[22] S. H. Rodger, "Using hands-on visualizations to teach computer science from beginning courses to advanced courses", in Proc. of the 2nd Program Visualization Workshop, 2002, pp: 103-112.

[23] R. B. Levy, M. Ben-Ari, and P. A. Uronen, "The Jeliot 2000 program animation system", Computers & Education, vol. 40, 2003, pp: 15-21.

[24] M. C. Carlisle, T. Wilson, J. Humphries, and S. Hadfield, "RAPTOR: A Visual Programming Environment for Teaching Algorithmic Problem Solving", in Proc. of 36th SIGCSE Technical Symposium on Computer Science Education, 2005, pp: 176-180.

[25] E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier. "Choregraphe: a Graphical Tool for Humanoid Robot Programming". In Proc. IEEE International Symposium, 2009, pp: 46-51.

[26] H. A. Ramadhan, F. Deek, and K. Shihab, "Incorporating software visualization in the design of intelligent diagnosis systems for user programming", Artificial Intelligence Review, vol. 16, 2001, pp: 61-84.

[27] J. Anderson and B. Reiser, "The LISP Tutor", Byte, vol. 10, 1985, pp: 159-175.

[28] RamalinghamV, LaBelle D, Weidenbeck S, "Self-Efficacy and Mental Models in Learning to Program", SIGCSE Bulletin Volume 36 Issue 3, ACM Press, NewYork – NY –US, 2004, pp: 171-175.

[29] Winslow L., Programming Pedagogy –"A Psychological Overview", SIGCSE Bulletin – Volume 28 Issue 3, ACM Press, New York USA, 1996, pp: 17-22.

[30] Westphal B, Harris F and Fadali M, "Graphical Programming: A Vehicle for Teaching Computer Problem Solving", 33rd ASEE/IEEE Frontiers in Education Conference, IEEE, Boulder Colorado, 2003, pp: 19-23

[31] Ben-Bassat Levy R, Ben Ari M and Uronen P, "An Extended Experiment with Jeliot 2000", In Proceedings of the First International Program Visualization Workshop, University of Joensuu Press, Porvoo Finland, 2001, pp: 131-140

[32] Stanford online parser, http://nlp.stanford.edu:8080/parser/, April 10, 2014.

[33] Edraw Visualization Solutions, http://www.edrawsoft.com/flowchart-examples.php, June 12, 2014.

[34] NIELSEN, J., Usability 101: Introduction to Usability, Nielsen Norman Group, Freemont CA - USA,[Accessed 06/05/2014] http://www.useit.com/alertbox/20030825.html.

[35] Warnlulasooriya, R., Palazzo, D. & Pritchard, D., Journal of Experimental Analysis of Behaviour, 88,1,Society for the Experimental Analysis of Behaviour, Bloomington - IA - USA, 2007, pp: 103-113.

[36] Hearrington., Learning Efficiency and Efficacy in a Multi-User Virtual Environment, National Educational Computing Conference, Eugene - OR - USA, International Society for Technology in Education, 2009.

[37] Cronbach, L. J., Coefficient alpha and the internal structure of tests. Psychometrika. Vol. 16, 1951, pp: 297-334.