JEPPY: An Interactive Pedagogical Agent to Aid Novice Programmers in Correcting Syntax Errors

Julieto E. Perez[1], Dante D. Dinawanao[2], Emily S. Tabanao[3]
Department of Computer Science, College of Computer Studies
MSU–Iligan Institute of Technology, Iligan City Philippines

***Abstract*—Programming is a complicated task and correcting syntax error is just one among the many tasks that makes it difficult. Error messages produced by the compiler allow novice learners to know their errors. However, these messages are puzzling, and most of the times misleading due to cascading of errors, which can be detrimental to running a syntax-error free program. In most laboratory setting, it is the role of the teachers to assist their students while doing activities. However, in our experienced, considering the large number of students in a class, it may seem difficult for teachers to assist their students one-by-one given the time constraints. In this paper, the design and implementation of an interactive pedagogical agent named JEPPY is presented. It is intended to assist novice learners learning to program using C++ as a programming language. In order to see on how students struggle or progress in dealing with errors, the proponents implemented the Error Quotient (EQ) developed by Jadud. The principles of the cognitive requirements of an agent-based learning environment were followed. The agent was put into test by novice learners in a laboratory setting. Logs of interaction between the embodied agent and the participants were recorded, aside from the compile errors and edit actions. These mechanisms show us some insight on the interaction behavior of learner to the agent.**

Keywords*—Pedagogical agent; error quotient; syntax-error correction; compile errors; human computer interaction

I. Introduction

Computer programming is a complicated task. According to Jenkins [1], Programming "is a complicated business" which requires the mastery of several skills such as problem solving, abstraction, mathematical logic and testing, debugging and so forth. In addition to this, in case of novice programmers, knowledge was found to be limited and shallow hence they lack the ability to write syntactically-correct programs. Over the years, several studies have been conducted to look at how compiler errors have affected the learning curve of students learning to program, particularly, novice programmers. Becker [2] showed that compiler errors can be frustrating and students in his study described them as "barriers to progress". In addition, Denny, Luxton-Reilly and Tempero [3] showed that students have difficulties locating and correcting syntax errors using average compiler. Moreover, Kummerfeld and Kay [4] concluded that even the more experienced students took significant time to correct some syntax errors. Studies have been conducted to understand interaction of learners to the compilers. Separate studies of Jadud [5] and Becker [6] showed a metric in quantifying these repeated errors. Jadud [5] called the interaction of the learners to the compiler as "compilation behaviour" and called the metric as the Error Quotient. To support learners in dealing cryptic messages produced by compiler, Ahmed, Kumar, Karkare, Kar, and Gulwani [7] developed a system called TRACER (Targeted RepAir of Compilation Errors) that perform repairs on compilation errors. In a study of Becker, Goslin, and Glanville [8], an enhancement to JAVA compile error messages was made and employed for intervention. Comparison between control and intervention groups showed that enhancing compiler messages is of advantage.

For environment of practice by novice programmers such as that in a laboratory setting, the current methods and tools employed focused on identifying behaviors using the online protocols and browser to inform teachers who among their students struggles and then provide manual intervention if necessary. However, considering the number of students in a classroom it is not realistic that the teacher can always assist the entire class for every laboratory session given some time constraints. This issue motivated the researchers in this study to come up another approach to augment the problem.

This study made an attempt in employing an embodied agent and see its potential use to aid novice programmers in their battle over syntax errors. This can help mentors attend to several other skills to consider in teaching programming, rather than focusing on assisting compile errors correction. However, skills like problem solving and logical reasoning were not yet addressed in this study and learning on that aspects requires different measures to help novice learners.

In this paper, the proponents presented the design and implementation of an interactive pedagogical agent which will be used as a tool to assist novice programmers in the daunting task of correcting syntax error produced by the compiler. Moreover, the proponents look into the interaction of the learner to the agent along with their interaction to the compiler. This can give us insights on the improvement of the agent and to the target benefit at which the agent was employed.

II. Review of Related Work

There were studies that looked at how novice learners interact with the compiler while practicing programming. Jadud [9] define novice compilation behavior as the study of students' interaction with their compiler while learning to program. In his study, Jadud [3] developed a quantification of the student's compilation behavior based on grounded theory. He called it the error quotient or EQ. Every record in the data logs represents one compilation event. Stored in each record is

the error message if there was an error at the time of compilation, the location of the error in the file which is reported by the compiler as a line number, and the source code. An EQ score is of the range 0 to 1.0, where 0 is a perfect score. An EQ score of 0 does not mean that the student made no syntax errors in their programming process. What it means is that at no point did the student encounter the same syntax error consecutively. Whereas a session scoring 1.0 means that every compilation resulted to the same syntax error all the time.

Agapito and Rodrigo [9] looked into students' compilation behaviors as they wrote their programs in C++ by analyzing automatically collected online protocols. Students' data were analyzed by computing for their Error Quotient. Results confirmed that freshmen programmers do experience difficulties and that the Error Quotient is a practical tool that can be used to characterize their compilation behaviors.

Many of the programming environment or IDEs used today have embedded capabilities or features added to help programmers do their job easily instead of just writing it using plain text editors. This same IDE is also used by novice programmers in their first programming experience using specific language. Many works reported development of automated syntax error correction. However, the approach does not care whether learners have assisted their own mistake.

Some works produced feedback through an interface where support is provided. Carter [10] developed an intelligent tutor for debugging called ITS-Debug. This is achieved by developing a system with four standard modules (Domain, Student, Pedagogical, Communication) of Intelligent Tutoring System. A web-based system was developed wherein students learn debugging through different phases. Students were able to edit the code, compile and run the code, and receive assistance on a host of syntax, runtime, and logical defects that might be present in the exercise or that they may inadvertently create themselves. In the study of Kummerfeld and Kay [4], a web-based reference guide was developed which catalogues some common C++ compiler generated errors.

So far, the work of Edwards, Rajagopal, Kandru [11] reported the use of embodied characters that assist learners in dealing syntax. The proponents developed an emotionally-intelligent pedagogical agents to deliver effective and efficient feedback to students about their programming assignments and also act as a teaching assistant for any general programming related queries. The main objective of their study is to communicate clearly the feedback about student programs while motivating them to perform better. This is so far, the work that was closely related in this study.

Veletsianos and Russell [12] defined pedagogical agents as anthropomorphous virtual characters employed in online learning environments to serve various instructional goals. Pedagogical agents were employed by Carlotto and Jacques [13], Kim [14], Liew, Zin, and Sahari [15], and Kim, Thayne, and Wei [16] in a form of an animated characters, virtual or digital characters. It was used as a chatbot as reported by Savin-Baden, Tombs, and Bhakta [17], an influencer such as of Kim and Baylor [18], or a tutor Kim [14]. They can also simulate conversations and nonverbal behavior according to Liew and Tan [19]. In the work of Schroeder, Romine, and Craig [20], pedagogical agent was employed to enhance student learning. Johnson and Lester [21] cited a nonverbal feedback capability of pedagogical agents. The nonverbal cues can take various forms including nodding or shaking the head, facial expressions such as smiling or surprise. This paper employs the use of nonverbal cues for the embodied agent and used the agent as an assistant.

III. Methodology

A. Defining Agent Design Requirements

According to Baylor [22], the prime cognitive consideration in the design of agent-based learning environment is the management of control. The first dimension of control involves instantiating the instructional purpose of the environment on a constructivist (high learner control) to instructivist's (high program/agent control) continuum. A critical issue from a constructivist approach to agent-based learning environments is in moderating between the agents taking over thinking for the student with the agent training the student to think more effectively. In the constructivist approach, the agent is a medium that does not teach the student directly. In this study, the presentation of knowledge about errors comes in a form of recall and example. Note that in an error message, the compiler may refer to some token in the code. Meaning, different token may appear even for the same error type. For example, the error message "expected ';' before 'int'" contains the token ';' and 'int' enclosed within single quote. In recall, the content presented by the agent will not specifically tell the student the specific solution but instead present the similar or general case. For instance, for the error mentioned above, the agent would say "Remember that in C++ every statement must end with a semi- colon. In an example, the agent would present an example statement with a semi-colon at the end. This is how the proponents push the student to do the thinking. The second dimension of control entails managing feedback, and several issues need to be considered: type, timing, amount, explicitness, and learner control of agent feedback. An important consideration in terms of feedback is that the pedagogical agent should not provide too many insights and thereby annoy the student. In the current design of intervention, the agent will depend on the current computed value of the error quotient. This means that whenever the student is stuck in a specific error, the agent will intercept every compilation. Although by default help should be minimal, part of our intention is to give us insight on the interaction of the learner to the agent in the environment. So, the proponents allow the agent to be proactively intervening as long as the EQ limit of 0.3 or greater was reached. Third consideration is when agent versus learner control is further defined through the desired relationship of the learner to agent. Some examples of instantiating the learner-agent relationship include the agent as learning companion, agent as mentor, multiple pedagogical agents, agent as personal assistant, or agent as resource. In this paper, the proponents define the role of the agent to be an assistant that informs the learner on their mistake. The feedback flows from agent looking at the error quotient and appears when EQ is greater than the value 0.3. The agent looks only on the first error message per compilation since the first error most of the time is the cause of cascading error and if not eliminated will cause the student to get stuck on that error. This is also

consistent to the existing computation of error quotient in which only one error was considered for computation in every compilation. Fourth, to be instructionally effective, the agent must assert enough control so that the learner develops confidence in the agent in terms of believability, competence, and trust. The critical issue that concerns believability is the message that the agent will provide. Incorrect message to provide will decrease trust and competence. The persona and behavior of the agent was also considered to make the agent believable.

B. The Agent Persona

Fig. 1 shows the Agents' gestures. Sequencing these gestures make up some form of behavior. The choice of the interface is a cartoon character and was named JEPPY. The proponents choose not a very serious character to capture the attention of the serious learners. The behavior space includes deictic and affective gestures as shown in the Fig. 1. The following gestures were combined to form actions that make the embodied agent more life-like.

Fig. 1. Deictic and Affective Gestures of JEPPY.

C. Testing

To test and validate the functionality of the components, the proponent put JEPPY in to a test with participants in an actual laboratory session. Participants were students taking up introductory programming course in a State University. Before the participants continue in the task, they were given questionnaire to verify whether they are really novice programmers. This is because the agent is intended for novice programmers only. There are 18 participants which where identified to be novice programmers. They were given a source code which contains cascading errors. Meaning, one error may come after another after correcting the first one. They were all given the freedom and time to finish the problem without asking help from other participants or instructor around.

D. The Architectural Design

The implementation follows the typical architecture of a pedagogical agent but was contextualized according to purpose of used. Fig. 2 shows the architecture of the agent in this study. The pedagogical module was implemented as plugin in Code::Blocks. The errors produced by the compiler were preprocessed to include only necessary information. The event logger was responsible on logging the preprocessed compiler errors, the edits done in the code, the interaction of the learner with the agent and the calculated value of error quotient. These data logged by the logger were inserted in the SQLite database. The communication module implemented using Java comprises the interface and inference controller. The interface is where the learner interacts with the agent. The embodiments are gif files which are retrieved depending on the interaction and current state of the learner.

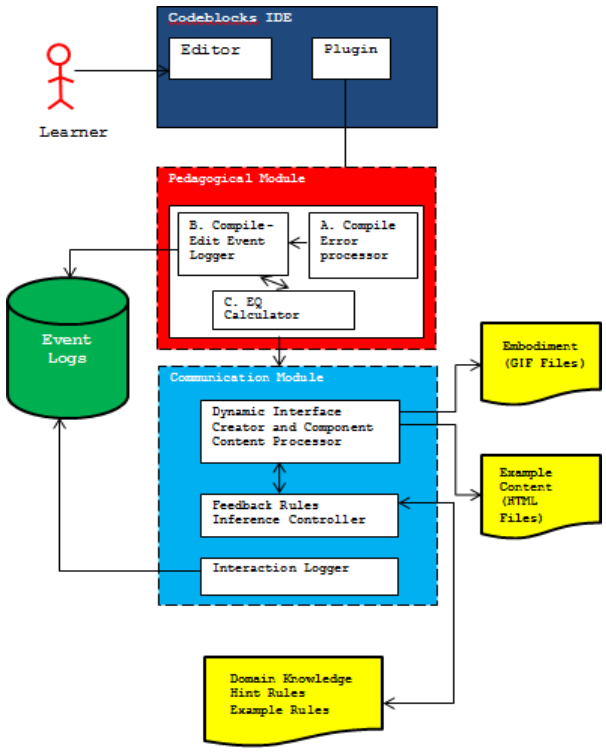

Fig. 2. Architectural Design.

Recalls which can be interchangeably call as hint and examples were written as an html files, which can then be viewed in the interface. The interface contains a balloon tip which is an open source program written in java. Html files which are retrieved from the domain module were displayed inside the balloon tip. The inference controller is responsible for retrieving knowledge during intervention. This part of the implementation connects the pedagogical module and the domain module. The knowledge on the errors was written in CLIPS as rules in an if-then format.

IV. Results and Discussions

A. The Implemented Agent

In the first compilation, the agent would appear and introduce itself to the student. Starting from the first compilation also, the logger is activated. So, every time the learner edits some lines in the code it will be recorded line per line. For every compilation starting from the third compilation, the two pairs of events can be created. At this point the Error Quotient can be calculated. When EQ is more than the threshold value, the agent will capture the first error, preprocessed it and retrieve message from the rules in the domain module that matches the error, and then display the help message through the embodied agent.

Table I shows an example EQ computation extracted from the compile-edit log. As per algorithm, the task is to compare

two successive compilations. For instance, from Table I, looking at compilation number 2 and 3, both compilations ended with error, so a penalty of 2 was added. Since both compilations have same error type (expected token before token), a penalty of 3 was added. However, both compilations do not have same error location and line edit made, so no penalty was added. The total score for this pair (compilation 1 and 2) is 5. The total score was divided to 11, which is the highest possible score, and is now the normalized value 0.5556. The final error quotient for this pair is the average of the sum of all the normalized score in each pair, in the given example, it is 0.3889.

The implemented agent was shown in Fig. 3 to Fig 9. Fig. 3 shows the appearance of the agent when it offers help from the learner. As one can see, the agent does not provide directly the help on the error identified. Instead, an option was given to see whether help is needed, or the learner already knows the error. When help is used, the agent will then provide the help as shown in Fig. 4. Fig. 5 shows the case wherein the error occurred again, and the agent will offer another help. Fig. 6 is the screenshot of the agent portraying like reading some notes when telling student to use example.

When help is used again, help will be provided in a form example as shown in Fig. 7. Fig. 8 and Fig. 9 are the affective gestures of JEPPY when it is sad and glad, respectively.

B. Result of Interaction based on the Logs

One critical part among the components is the correct message or support that the agent will provide. The interaction log provides a way for us to see whether correct help is given to an error message. Recorded in every row was the error message which is a result in preprocessing stage during compilation. Also, in the same row, was the help coming from the domain knowledge which is a result of the inference engine.

Aside from validating the functionality of the components through the logs, it also gives us some observations on the interaction of the learner to the agent. Out of 538 times that the agent appears, only 159 or 29.55% of the time the agent was used. It can also be observed that there are 119 or 22.12% of the time the agent was closed when help is asked. The proponents can also see instances wherein there is no interaction in an intervention, meaning the agent was ignored and after 20 seconds without any interaction it pops out. There are 260 or 48.33% of the time that the agent was ignored. The large number of time that the agent was ignored by students is maybe because they were so engaged in attempting to correct error by themselves. As mentioned by Jadud [3], students took significant time editing and compiling their code, and after several attempts without success, they may fall into frustration. But here, with the presence of JEPPY, we can be able to prevent such case. We can see that in the sequence of usage. From 159 interventions, 106 or 66.7% were hint usage and 53 or 33.33% were example usage. Even the students are proactively debugging these errors by themselves and do not use help even when they need it, based on the logs, out of the 106 hints usage, 70 or 66.04% of the time wherein errors were encountered are corrected after using hint. When error was not eradicated, the agent can reinforce this by offering an example. We see that there are 11 instances in the total usage wherein hint is immediately followed by example and the error was corrected after it. There is a total of 81 or 76.42% of errors corrected after using the support provided. In case of example usage only, meaning not preceded with hint, there is 56.60% of the total usage wherein the error was corrected right after.

Although the figures presented are not at large, the potential of JEPPY can be seen in helping the novice learners in dealing syntax error, of course, with further improvement.

TABLE. I. SAMPLE ERROR QUOTIENT CALCULATION

Compilation no.	**Error message**	**Error message type**	**Error location**	**Both event - s end with error**	**Same error type**	**Same error location**	**Same edit location - on**	**Pair no**	**score**	**Normal-ized score**	**Sum of normali -zed score**	**Error quotient**
1	‘ans’ was not declared in this scope	error was not declared in this scope	55									
2	Expected ‘while’ before ‘cin’	expected token before token	29	2	0	0	0	1	2	0.2222	0.2222	
3	Expected ‘;’ before ‘endl’	expected token before token	50	2	3	0	0	2	5	0.5556	0.7778	0.3889
4	Expected ‘}’ before ‘else’	expected token before token	52	2	3	0	1	3	6	0.6667	1.4444	0.4815

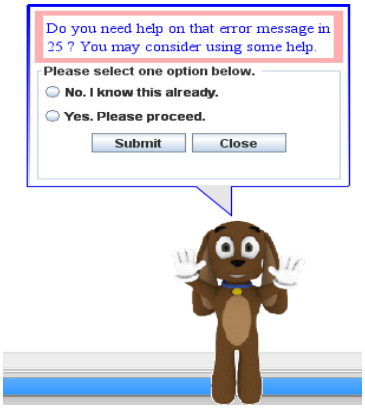

Fig. 3. JEPPY Offering Help through Hint.

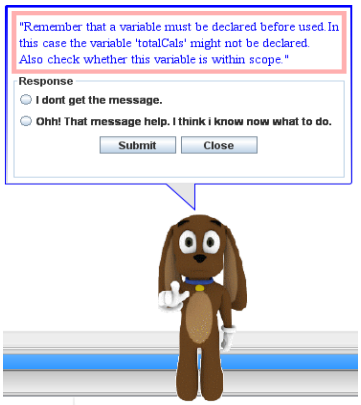

Fig. 4. JEPPY Showing Hint.

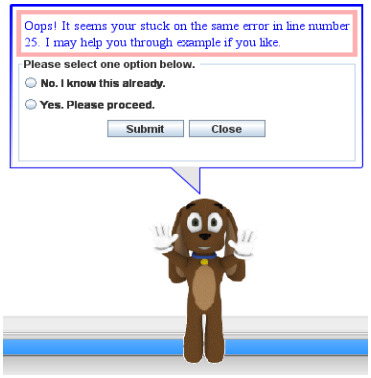

Fig. 5. JEPPY Offering Help through an Example.

Fig. 6. JEPPY when Instructing to Read Help Carefully.

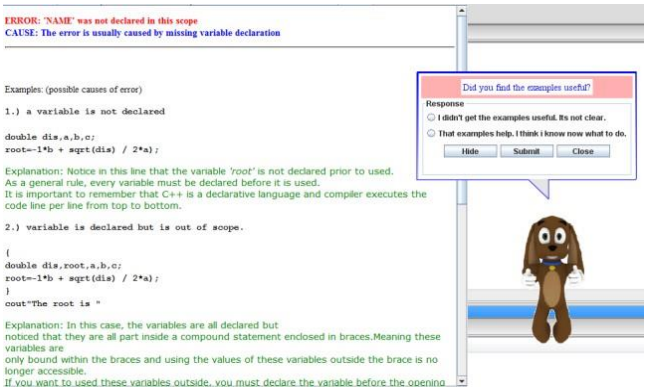

Fig. 7. JEPPY Showing Example to an Error.

Fig. 8. JEPPY when Help was Ignored or not used.

Fig. 9. JEPPY when Help was used and Error was Corrected.

To see whether content in the support is helpful, part of the interaction by the agent is to ask the learner whether the message is clear or helpful. There are 66 or 41.51% instances wherein student responded on the question whether hint is clear or understandable. From the total responses, all 66 of it responded that the message is clear. For the example usage, however, there is only one response which said that the message is clear.

The summary of our logs had given us insight in terms of interaction. In the design, the agent was intended to be proactive by having smaller threshold value of Error Quotient. But our logs tell us that the agent must be designed to carefully select

timing in intervention, otherwise, the learner might get annoyed. Probably models on interaction along with EQ should be developed for timing in intervention. Nevertheless, when help is being used, the agent can be of help before the learner falls into frustration. However, it should be noted that our logging mechanism was not intended to deeply look on the efficacy of learning. The logs enable us to verify and validate the functionality of every component and give us opportunity to gain insight for further improvement of the agent.

V. Conclusion and Future Work

In this paper, the researchers presented the design and implementation of an interactive pedagogical agent. It was successfully embedded as a plugin in an Integrated Development Environment named Code::Blocks. The said environment for developing real-world applications was also used by the students in our institution. However, it was not developed to care on the problem encountered by Novice programmers such as syntax error correction. Hence, through this work, the researchers were able to address one of the many problems a Novice programmer may encounter.

Although our domain is specific to C++ as programming language, the modular fashion of the architectural design on the components can be easily expanded. For instance, rules containing errors and their corresponding help or corrections can be added without any changes in the rule engine as long as it conforms to the pattern. Currently, the study does not include yet the evaluation on the learning gain. It can be seen, however, that by using the computed EQ, one can determine how well a student is progressing with or without JEPPY. This can be done with a large number of participants and an ample time. The current work done focuses on the design and implementation of the agent and the EQ.

References

[1] T. Jenkins, “On the difficulty of learning to program,” 3rd Annual LTSN -ICS Conference. University of Ulster, LTSN Centre for Information and Computer Sciences, 2002.

[2] B.A. Becker, “An effective approach to enhancing compiler error messages,” In Proceedings of the 47th ACM Technical Symposium on ComputingScience Education, 126–131, 2016.

[3] P. Denny, A. Luxton-Reilly, E. Tempero, J. Hendrick, “Understanding the syntax barrier for novices,” ITiCSE In Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education, 208–212, 2011.

[4] S. K. Kummerfeld and J. Kay, “The neglected battle fields of syntax errors,” In Proc. Fifth Australasian Computing Education Conference, 105-111, 2003.

[5] M.C. Jadud, “Methods and tools for exploring novice compilation behaviour,” Proceedings of the 2006 international workshop on Computing education research, pp. 73-84, 2006.

[6] B. A. Becker. “A new metric to quantify repeated compiler errors for novice programmers,” In Proceedings of the 21st ACM Conference on Innovationand Technology in Computer Science Education, pp. 296–301, 2016.

[7] U. Z. Ahmed, P. Kumar, A. Karkare, P. Kar, and S. Gulwani, “Compilation error repair: for the student programs, from the student programs,” In Proceedings of the 40th International Conference on SoftwareEngineering: Software Engineering Education and Training, 78–87, 2018.

[8] B. A. Becker, K. Goslin, and G. Glanville, “The effects of enhanced compiler error messages on a syntax error debugging test,” In Proceedings of the 49th ACM Technical Symposium on Computer Science Education, 2018.

[9] J. L. Agapito, M. M. T. Rodrigo, “An analysis of novice programmers' compilation behaviors in c++,” Philippine Information Technology Journal, 2012.

[10] E. A. Carter, "An intelligent debugging tutor for novice computer science students," Theses and Dissertations, 2014.

[11] S. H. Edwards, M. B.M. M. Rajagopal, N. Kandru, “Pedagogical agent as a teaching assistant for programming assignments: (abstract only),” Proceedings of the 49th ACM Technical Symposium on Computer Science Education, p.1079, February 2018.

[12] G. Veletsianos, G. Russell, “Pedagogical agents,” handbook of research on educational communications and technology, 4th Edition, pp. 759-769, 2014.

[13] T. Carlotto and P. A. Jaques, “The effects of animated pedagogical agents in an english-as-a-foreign-language learning environment,” International Journal of Human Computer Studies, vol 95, pp.15–26, 2016.

[14] Y. Kim, “The role of agent age and gender for middle-grade girls,” Computers in the Schools, vol 33, pp. 59–70, 2016.

[15] W. T. Liew, N. A. M. Zin, and N. Sahari, “Exploring the affective, motivational and cognitive effects of pedagogical agent enthusiasm in a multimedia learning environment,” Human-Centric Computing and Information Sciences, 7(9), 2017.

[16] Y. Kim, J. Thayne, and Q. Wei, “ An embodied agent helps anxious students in mathematics learning,” Educational Technology Research and Development, 65(1), 219–235, 2017.

[17] M. Savin-Baden, G. Tombs, and R. Bhakta, “Beyond robotic wastelands of time: abandoned pedagogical agents and new pedalled pedagogies,” E-Learning and Digital Media, 12(3-4), 295–314, 2015.

[18] Y. Kim, A. L. Baylor, Pedagogical agents as social models to influence learner attitudes,” Educational Technology, 47(1), 23–28, 2007.

[19] W. T. Liew, and S. M. Tan, “Virtual agents with personality: adaptation of learner-agent personality in a virtual learning environment,” 11th International Conference on Digital Information Management, pp. 157–162, 2016.

[20] L. N. Schroeder, W. D. Romine., and S. D. Craig, “Measuring pedagogical agent persona and the influence of agent persona on learning,” Computers and Education, 109, 176–186, 2017.

[21] W.L. Johnson, J.C Lester, “Face-to-face interaction with pedagogical agents, twenty years later” International Journal of Artificial Intelligence in Education, 26(1), 25–36, 2016.

[22] A. Baylor, "Cognitive requirements for agent-based learning environments,", Proceedings of the 2001 International Conference on Advanced Learning Technologies, pp. 462-463, 2001.