Avrupa Bilim ve Teknoloji Dergisi
Özel Sayı 29, S. 415-424, Aralık 2021

Araştırma Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
Special Issue 29, pp. 415-424, December 2021

Research Article

Conceptual Design of Python IDE with Embedded Turkish Spoken Chatbot that Analyzes and Corrects the Syntax Errors

Turgay Tugay Bilgin[1*], Erdem Yavuz[2]

[1*] Bursa Teknik Üniversitesi, Müh. ve Doğa Bil. Fak., Bilgisayar Mühendisliği Bölümü, Bursa, Türkiye (ORCID: 0000-0002-9245-5728), turgay.bilgin@btu.edu.edu.tr
[2] Bursa Teknik Üniversitesi, Müh. ve Doğa Bil. Fak., Bilgisayar Mühendisliği Bölümü, Bursa, Türkiye (ORCID: 0000-0002-3159-2497), erdem.yavuz@btu.edu.tr

(International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT) 2021 – 21-23 October 2021)

(**DOI:** 10.31590/ejosat.1035421)

ATIF/REFERENCE: Bilgin, T.T. & Yavuz, E. (2021). Conceptual Design of Python IDE with Embedded Turkish Spoken Chatbot that Analyzes and Corrects the Syntax Errors. *European Journal of Science and Technology*, (29), 415-424.

Abstract

Intelligent agents, in addition to act as a cognitive tool, they can also to be designed as learning agents. In this way, they can learn the user's behavior from users' past behaviors. Learning agents can analyze the user's / student's behavior to a task, build a database of past activities, and suggest better strategies. This study provides a detailed review of interactive guidance using intelligent agents, and then introduces a concept model for a Conversation-based Turkish Python Integrated development environment (IDE) that can analyze user syntax errors and help them to correct errors. The model proposed in this study consists of three layers. These are user interface layer, middle layer, and Python interpreter layers. User interface layer consists of code editor and chatbot components. Middle layer includes code structural control subsystem, code error manager, and intelligent agent subsystems. The code structural control module analyzes conditions, loops, branching, and other types of program flow controls in the user's code. The code error manager analyzes the error outputs of the user code which is generated by the Python interpreter. The intelligent learner, on the other hand, uses these inferences to understand the reason for the student's error and convey the necessary actions to the user by the help of chatbot and suggest possible corrections. The proposed Integrated development environment (IDE) has a well-designed UI that can be easily adapted by newcomers. The coding editor can be used as a stand-alone desktop software, or it can also connect to a cloud storage to store user codes in the cloud. A web application is also planned for our proposed IDE. The teacher will be able to assign homework to the student over the web. The student will be able to view these assignments on the web, do the assignments in the desktop editor and send them back to the teacher over the web. In addition, each user's error characteristics will be analyzed, the success of the learning will be measured, and the deficiencies of the students will be determined using the Intelligent Agent Subsystem.

Keywords: Integrated Development Environments, Chatbots, Intelligent Agents.

Sözdizimi Hatalarını Analiz Eden ve Düzelten Türkçe Sohbet Robotuna sahip Python Tümleşik Geliştirme Ortamı Kavramsal Tasarımı

Öz

Zeki etmenler, bir bilişsel araç olarak hizmet etmenin yanı sıra, kullanıcıların geçmişteki davranışlarından hareketle kullanıcının davranış biçimini öğrenebilecek şekilde, yani öğrenen etmen olarak tasarlanabilirler. Bu şekilde tasarlanmış etmenler kullanıcının/öğrencinin bir göreve yaklaşımını analiz edebilir, geçmiş faaliyetler için bir veri tabanı oluşturabilir ve daha iyi stratejiler önerebilir. Bu çalışma, zeki etmenler ile etkileşimli yönlendirme çalışmaları hakkında detaylı bir inceleme sunmakta ve ardından kullanıcı hatalarını analiz edebilen diyalog tabanlı Türkçe Python kod editörü tasarımı için bir konsept model ortaya koymaktadır. Bu çalışmada önerilen sistem, üç katmanlı mimariye dayandırılmaktadır. Bu katmanlar kullanıcı arayüzü katmanı, orta katman ve Python yorumlayıcısı katmanlarıdır. Kullanıcı arayüzü katmanı; editör ve sohbet robotu bileşenlerinden oluşmaktadır. Orta katman; kod yapısal kontrol, kod hata yöneticisi ve öğrenen zeki etmen alt sistemlerini içerir. Kod yapısal kontrol modülü kullanıcının kodundaki koşul, döngü, dallanma ve diğer tür program akış kontrollerinin analizini yapar. Kod hata yöneticisi, kullanıcının yazdığı Python kodunun, Python yorumlayıcı tarafından çalıştırılması sonucunda elde edilen hata bildirimlerini analiz eder. Öğrenen zeki etmen ise bu çıkarımları kullanarak öğrencinin hatasının sebebini anlayarak bunları düzeltmesi için gereken işlemleri sohbet robotu aracılığıyla kullanıcıya

[*] Corresponding Author: turgay.bilgin@btu.edu.edu.tr

aktarır ve olası düzeltmeler önerir. Önerilen geliştirme ortamının, kodlamaya yeni başlayanların kolay adapte olabileceği ergonomiye sahip olması planlanmıştır. Bu amaçla kullanıcıya yönlendirme sağlamak için diyalog tabanlı etmen içeren bir alt sistemin düşünülmüştür. Kodlama editörü tek başına bir masaüstü yazılım olarak kullanılabileceği gibi, yazılan kodları bulut ortamında depolama özelliğine de sahip olabilmektedir. Bulut ortamında eğitici/öğretmen tarafından kullanıcıya atanan ödevleri alabilme ve öğrencilerin çözümlerini tekrar eğiticiye gönderebilme özelliklerine sahip olması planlanmıştır. Ayrıca, her bir kullanıcının hata analizlerinin yapılabilmesine olanak sağlayarak öğrenmenin ne ölçüde gerçekleştiği ve öğrencilerin hangi konularda eksiklerinin olduğunun görülebilmesine olanak sağlayan bileşenler tasarlanmıştır.

Anahtar Kelimeler: Tümleşik Geliştirme Ortamları, Sohbet Robotları, Zeki Etmenler.

1. Introduction

One of the earliest studies on the use of intelligent agents for educational purposes was carried out by Roesler and Hawkins in 1994. In their study, the most general definition of intelligent agents is expressed as software that can adapt to individual habits and try to assist users in routine computer tasks [1]. This technology requires using all methods such as artificial intelligence, natural language processing (NLP) and human-machine interface. In a study, Harmon classify the agents into three groups [2]. Harmon states that his classification is based on how the agents do, not what they do. These are end-user programmed (or simple) agents, knowledge-based systems (or smart) agents, and self-learning (or intelligent) agents. On the one hand, Russell and Norvig divides the factors into five classes according to their intelligence and ability levels [3]:

- *Simple Reflex Agents*
- *Model-based Reflex Agents*
- *Goal-based Agents*
- *Utility-based Agents*
- *Learning Agents*

In addition to serving as a cognitive tool, intelligent agents can also be designed (as a learning agent) to learn the user's behavior from the past behaviors of the users. Agents designed in this way can analyze the student's approach to a task, create a database of past activities, and suggest better strategies. One of the most comprehensive studies on intelligent agents supported by machine learning was published by Maes in 1997. Maes proposed a machine learning approach to develop the intelligent agent in the study [4]. In Maes' approach, the agent constantly improves its abilities over time. The agent is competent from four sources: (1) it realizes the behavior by tracking the user; (2) it provides feedback to the user directly or indirectly; (3) it can be trained with the examples provided by the user; (4) it may seek advice from the other agents who have helped others before for the same task.

One of the first studies suggesting that intelligent learning agents can be used in education was carried out by Kearsley in 1993 [5]. According to Kearsley in that review, intelligent agents can serve successfully as coaches or consultants. Kearsley stated that it is necessary to be able to create agents in a programming environment and students should be able to use them directly. In the approach, students will learn to command an agent and help the agent perform a complex task on its own. As Kearsley suggests, it offers a new paradigm based on the notion of intelligent agents, shared skills between humans and computers, and collaborative learning.

In the study published by Giraffa and Viccari in 1998 [6], it was stated that the agents used in intelligent learning environments (ILE) should be named pedagogical agents. Pedagogical agents are those that have a set of teaching goals and plans and the resources necessary to achieve these goals [6]. The same authors proposed a taxonomy classifying agents in another study and expressed the relationship of pedagogical agents with other agents in this taxonomy as seen in Figure 1 [7]. According to this taxonomy, pedagogical ones are both software-based and artificial intelligence.

2. Literature Review

According to the studies of Chen et al. [8], pedagogical agents can be created in different ways such as animated factors or text-based agents. [8]. Animated pedagogical agents are often the focal point the interface and software. Text-based pedagogical agents can be positioned in a fixed area on the user interface. These can provide users with hints and guidance, or they can be in the form of a chat window. In that study, Chen et al. designed, implemented and measured the impact of CoLeMo, a collaborative learning environment for UML (Unified Modeling Language). In the aforementioned study, two types of pedagogical agents were designed to facilitate collaboration and improve learning. These were expressed as a domain agent and a facilitator agent. Both of them provide text-based recommendations based on students' activities. The domain agent knows the rules for UML diagramming and is responsible for making recommendations, thanks to its knowledge of the domain [8]. For example, if a student tries to execute an operation against the UML rules, the domain agent explains why this operation is illegal and what the correct action is. The facilitator is responsible for making recommendations to coordinate collaboration. This agent monitors activities of each student and the whole group and makes recommendations for organizing participation and collaboration [8].

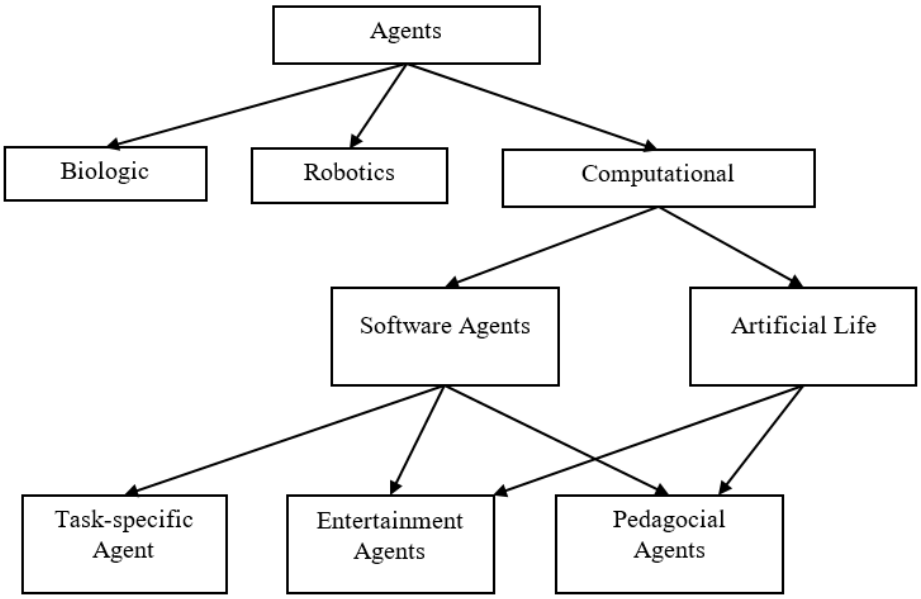

Figure 1. Taxonomy of agents according to Giraffa and Viccari [7].

Han et al. developed a peer-learning-based software in programming education using pedagogical agents in their study published in 2010 [9]. This system mimics the roles of "teacher" and "learner" in a peer-learning method from pedagogical and technical perspectives. The peer-learning agent uses the Bayesian network and artificial intelligence methods as well as teaching and learning methods. The role relationship between the peer-learning agent and the student is shown in Figure 2. This model, which combines peer-learning tools with a teaching and learning strategy, has been shown to be effective in having students gain programming skills [9].

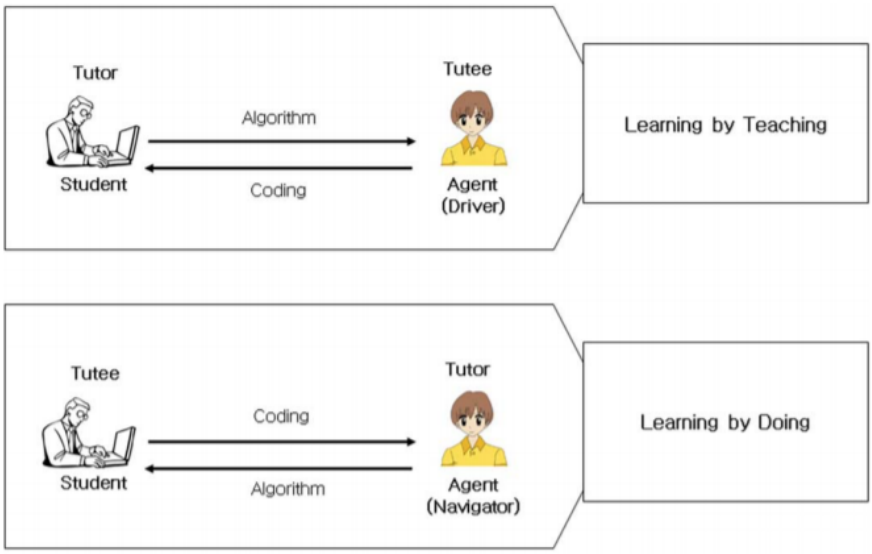

Figure 2. The relationship between the peer-learning agent and student [9].

Pedagogical agents can also be formed as conversational agents. Such agents imitate people's communication methods and often chat with the student(s) through speech, text messages, facial expressions, gestures, or other body language actions [10]. Having trailing the historical roots of pedagogical agents, the mentioned study focused on creating an agent that is both knowledgeable and that develops a social relationship with the learner. Until now, many conversational agents have been developed and they can successfully perform multiple pedagogical roles such as teacher, learning-peer or coach [11]. Furthermore, Walker et al. (2011) showed that having a speaker giving guiding directives/instructions during a peer activity can increase the conceptual depth of students' expressions [12].

Tegos et al. developed and used a prototype conversational agent system called MentorChat [13]. MentorChat is basically a

cloud-based computer-assisted collaborative learning system. This system also enables a teacher to structure and distribute some dialogue-based collaborative activities to students. The architecture of the system, whose user interface is shown in Figure 3, consists of three modules: student, teacher and speaker agents. The speaker agent of MentorChat uses an animated two-dimensional avatar representation (Figure 3C) and uses a text-to-speech engine to deliver instructions to the student. The agent shows their notifications in a popup (Figure 3D) and not in the common chat area (Figure 3B). In this way, the attention of the students is drawn to the last message of the agent and they are provided to respond to this message.

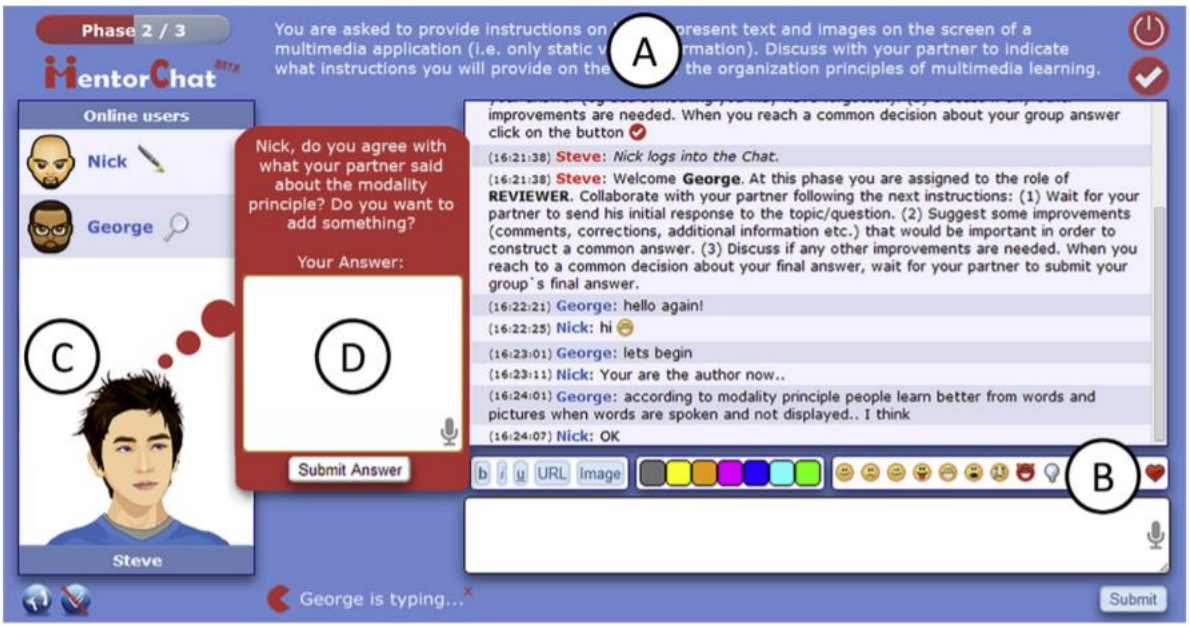

Figure 3. MentorChat user interface [13].

2.1. The use of chatbots in education

A chatbot is a computer program designed to simulate a chat with its users, usually over the Internet [14]. “Furthermore the analogy that a chatbot often treats a conversation like a game of tennis can be used to describe the conversation flow of a chatbot, i.e. get message, reply, get message, reply, and so on” (The Oxford Dictionary, 2018). Deryugina (2010) makes almost the same definition, but by adding the word “intelligent” before communication, it indicates the need for intelligent answers rather than just random answers. Chatbot is a general term for many other names such as Chatterbot, Conversational Agent, Intelligent Pedagogical Agent (IPA), and Conversational System and Pedagogical Tool.

Chatbots can be developed in many ways, but a popular and fairly simple way is to use Artificial Intelligence Markup Language (AIML) [14]. AIML, was built by Dr. Wallace and an open source community (Alice foundation) to work as the brainchild of the chatbot [15]. AIML is a derivative of XML and in this variant, AIML files as the knowledge base or brain of chatbots allow the person managing the chatbot to add information to the chatbot [14]. Chatbot A.L.I.C.E. was developed using AIML based on categories containing a stimulus or pattern, and a template in response. In this approach, the category patterns are then matched to find the most appropriate response to a user input.

Cleverbot was launched in 2008, and unlike other chatbots, its responses were not pre-programmed. Instead, an approach was adopted where a user would write a comment or question and learn directly from human input, where Cleverbot would find all keywords or an exact phrase matching the input [16].

Again, in a study published in 2008, it was reported that two chatbots (T-Bot and Q-Bot) were developed to teach and evaluate students. The authors have developed two AIML-based chatbots to assist students in the learning process and support teaching activities on an e-learning platform such as Claroline or Moodle. One of the chatbots plays the role of a teacher (T-bot) and communicates with the student in natural language to provide adequate and domain-specific answers to students and guide students to the correct course material. The other has the role of an evaluator (Q-bot) and can monitor and supervise the student through personalized questionnaires. Both chatbots have been developed as easily integrated modules for Claroline or Moodle [14].

A chatbot can be implemented as rule-based or artificial intelligence (AI) [6]. However, unlike rule-based chatbots, AI-based chatbots can become smarter and more scalable over time. Also, AI-based systems have recently become a popular choice for chatbot researchers. In this context, the recurrent neural network based sequence-to-sequence (Seq2Seq) model is reportedly one of the most researched models to implement the artificial intelligence chatbot and has shown great progress since its emergence in 2014 [17]. However, it is still open to progress and has not yet been widely implemented in educational chatbot development.

Chatbots used in education seem to promise to have a significant positive effect on learning achievement and student satisfaction [18]. For example, the University of Georgia has created a chatbot called "Jill Watson", based on IBM's Watson platform, developed specifically to process forum posts from students enrolled in a computer science course [18]. As a result of this, students were more interested in the lesson and stated that

they wanted them to have the same opportunity in other lessons [19].

Today, students have access to most of their information about courses and assignments online. Therefore, chat programs, such as NerdyBot [20], provide important support in learning processes [21]. NerdyBot is another educational chatbot available via Facebook Messenger that can perform school-related tasks such as solving math equations, drawing graphs, searching for definitions, and finding historical events [22]. The screenshot of NerdyBot is shown in Figure 4.

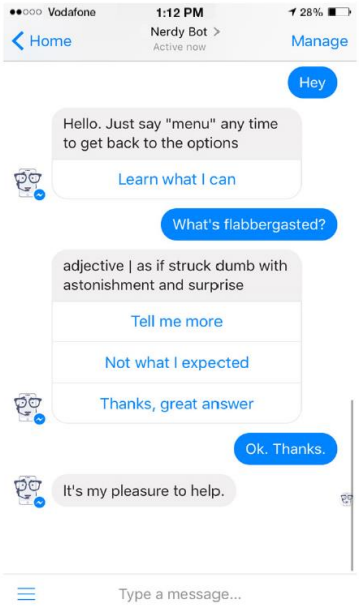

Figure 4. NerdyBot screenchot [20].

It is asserted in the study that such programs offer a communication medium where Generation Z are very familiar. These students tend to join virtual learning groups to chat with other students. Chatbot usage may make it possible reminders about exams to be sent, generative systems help to understand the curriculum. Continuing developments in NLP make it possible for systems to understand students' questions. These students are inclined to believe information from chatmates rather than search engines [22]. Therefore, the thesis that knowledge should be spread in an environment where students pass a lot of time can be defended.

3. Proposed Architecture

The system proposed in this study is designed with a three-layer architecture. Although the three-tier architecture is generally preferred in data-intensive applications, it has been adapted to the requirements of this study. The block diagram of the proposed three layers is shown in Figure 10.

These layers are:

a. User Interface Layer: The components in this layer are Editor and Chatbot.
b. Middleware Layer: The components in this layer are “Code Structure Checker”, “Code Error Manager”, "Learning Agent Subsystem".
c. Python Interpreter: This layer consists of Python interpreter itself, which will run the code developed using the Editor.

It is planned to develop the designed Python editor as a desktop application. However, in addition to this, a Cloud Storage module is planned so that the student codes can be stored in the cloud environment, and a web-based administration panel (Dashboard) is planned so that the teacher can monitor the statistics and the analysis about student errors and also teacher can assign homework to the student via this panel.

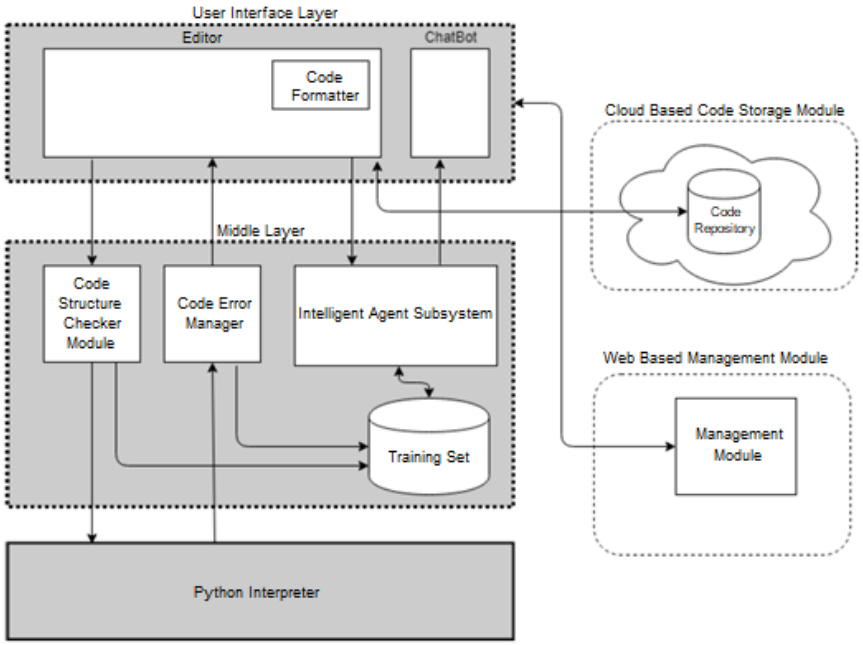

Figure 10. Block diagram of the proposed code editor.

3.1. User Interface Layer

An innovative design approach is planned while developing the coding editor interface. The sample GUI interface is given in Figure 11. The user interface is designed ergonomically so that junior programmers can learn it with minimum effort.

As can be seen in the figure, the editor screen has been designed to be as simple and uncomplicated as possible. Codelets enable some frequently used code-parts to be moved to the editor area by "drag-and-drop" are placed in the left section. Compiling, saving, debugging, etc. buttons are located at the top. The output of the written code is located just below the editor. The dialog/chatbot is docked on the right. In this part, the chatbot displays the dialog texts to the user and at the same time takes all the input from the user to the learning agent module which is located in the middle layer. The learning agent module is detailed in the next section. *Chatbot*, is designed to be able to ask questions to the student in some cases, and the student can choose the answer from the list given to her/him, as seen in the Chatbot dock in Figure 11. The "code formatter" component is responsible for indentation and syntax highlighting in the editor.

The Natural Language Toolkit (NLTK) library is used to design chatbot backend. NLTK, is a suite of libraries and programs for symbolic and statistical natural language processing (NLP) written in the Python programming language. It was developed by Steven Bird and Edward Loper in the Department of Computer and Information Science at the University of Pennsylvania [23]. The aforementioned study is available to users on the GitHub platform as open source [24]. Python and QT Framework library, HTML and JavaScript technologies were preferred in order to be able to develop platform independent Desktop Editor.

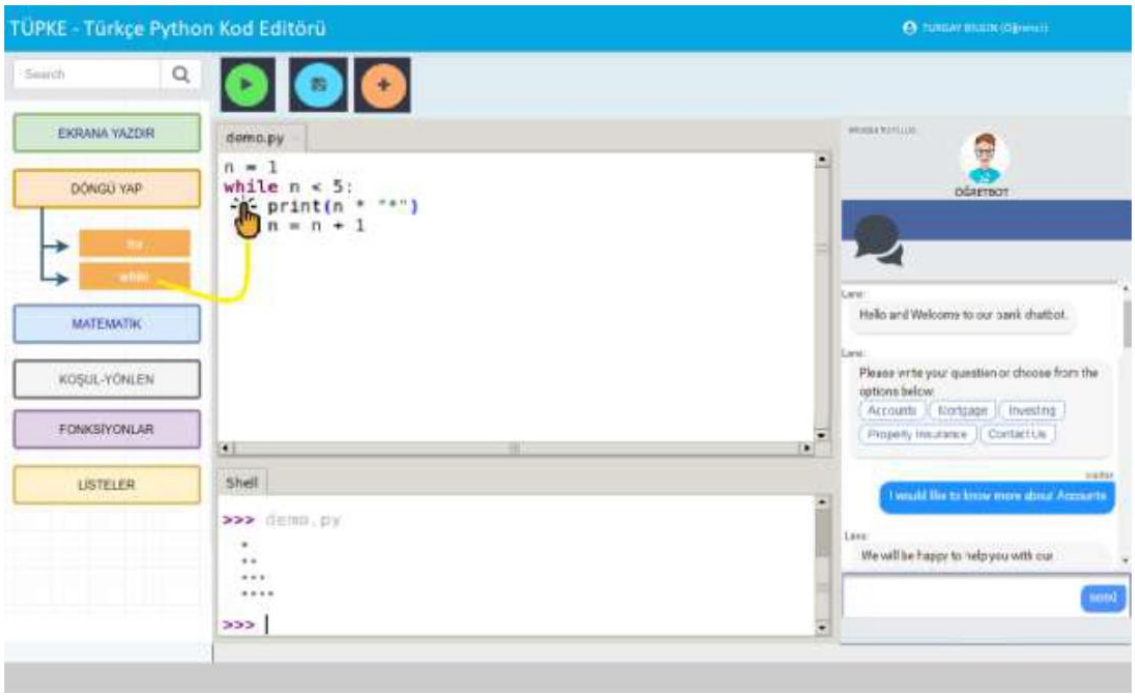

Figure 11. Proposed code editor GUI design.

3.2. Middle Layer

This layer consists of "Code structural control", "Code error manager" and "Intelligent Learning Agent Subsystem" units. "Code structural control" module checks the syntax of structures such as condition, loop, branching in the user's code and performs the task of checking compliance with code writing standards and formal analysis. Other libraries such as PyRight [25], Flake8 [26], pyflakes [27], PyChecker [28] can also be used for code structural analysis. These tools are known as “static code analysis tools”. They perform the formal analysis of the code written by the user without actually running it in the interpreter, and they can identify the syntax errors’ position and the definition of the error more accurately than the default error messages of the Python compiler. For example, let's assume that a "test.py" python code is written as in Figure 12.

```
print("Hello World!"
```

Figure 12. Sample Python code

The 3.X version python compilers display “SyntaxError: unexpected EOF while parsing” error when this piece of code is run. There is an incorrect use of the "print" function in this piece of code due to the missing closing parenthesis. On the other hand, static code analysis tools such as PyRight can detect such problems. Pyright is a fast type checker meant for large Python source bases. It is an open source project and mainly contributed

by Microsoft. A portion of the Python and PyRight output for this piece of code is given in Figure 13.

```
C:\>python test.py
  File "test.py", line 2
                    ^
SyntaxError: unexpected EOF while parsing

C:\>pyright test.py
Searching for source files
Found 1 source file
C:\test.py
  C:\test.py:1:21 - error: Expected ")"
1 error, 0 warnings, 0 infos
Completed in 0.626sec
```

Figure 13. Comparison of Python interpreter and PyRight outputs.

The error statement " test.py:1:21- error: Expected ")" ", which is taken from the PyRight output as shown in Figure 13, is sent to the Intelligent Learning Agent which is actually chatbot itself. Chatbot takes the position of error and the error definition, and generates a Conversational Dialog Tree in order to interact with the student. During the rendering of the English error texts, an offline lookup table based English-to-Turkish translator subsystem translates the error into the Turkish Language.

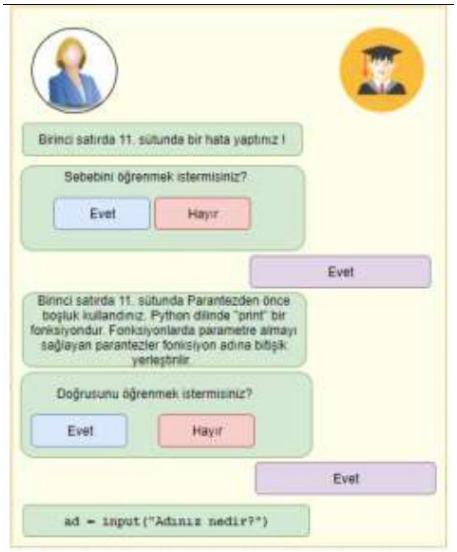

Figure 14. The example chatbot-student dialog tree.

Intelligent Agent subsystem generates Dialog Tree by using this information. A dialogue tree or conversation tree is a game mechanic used in many adventure games and role-playing video games. When interacting with a non-human character, the player is given a choice of what to say and makes subsequent choices until the conversation is over. An example computer generated dialog tree in Turkish Language is given in Figure 14. Format or syntax errors caught by “Code structural control” and “Code error manager” and violations of python coding standards are stored as relational database in the “training set” database. Thus, a knowledge base is generated for the learning agent.

Intelligent Learning subsystem tracks the user’s error event sequences. Figure 15 shows an example error event sequence. In the example sequence, the student wants to print “Hello” on the screen. Since he does not know the use of the” print" function, he cannot achieve this task.

```
>>> print merhaba
SyntaxError: Missing parentheses in call to 'print'. Did you mean print(merhaba)
?
>>> print(merhaba)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(merhaba)
NameError: name 'merhaba' is not defined
>>> print("merhaba)

SyntaxError: EOL while scanning string literal
>>> print(merhaba")

SyntaxError: EOL while scanning string literal
>>> print"merhaba)

SyntaxError: EOL while scanning string literal
>>>
```

Figure 15. The sample error event sequence.

In our proposed model, the first task is to generate “key:value” pairs database. The “key” is the code fragment that consists of syntax errors and the “value” are the “syntax error” messages generated by the python interpreter or python code linter. Then, any further syntax errors that may be occur due to the user modifications are also stored in the same format. Sequential patterns are generated by putting all the syntax errors in the form of a sequential list. These patterns are also recorded in the "training set" database.

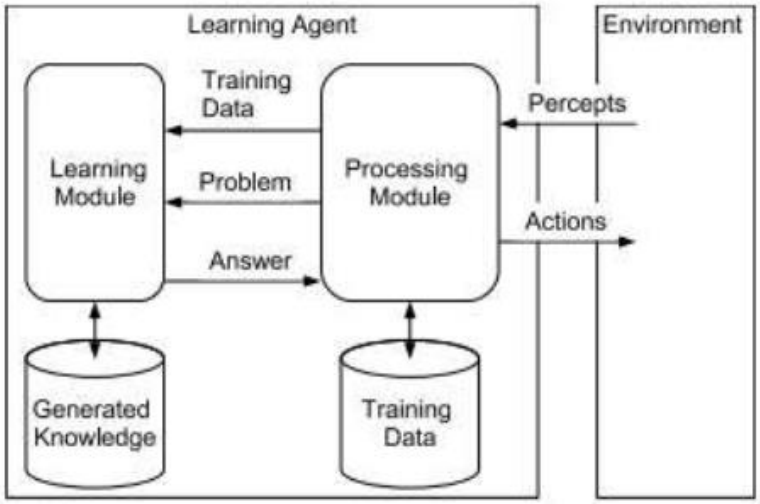

Figure 16. Proposed architectural block diagram for learning factor [29].

"Learning Intelligent Agent Subsystem", the architecture proposed by Śnieżyński and Bartlomiej in 2008 for "Learning Agents" [29] was modified and adopted to our proposed model. The block diagram of the architecture is given in Figure 16. The learning agent consists of four modules:

1. *Processing Module*: It is responsible for basic agent activities such as storing training data, executing the learning process, using the learned information, percept environment, and creating action.

2. *Learning Module*: It is responsible for executing the learning algorithm and answering the questions of the "Processing Module" regarding the use of the learned knowledge.

3. *Training Data*: It is the repository where the samples used for learning are stored.

4. *Generated Knowledge*: It is the repository where learned information is stored.

In the proposed architecture, the components interact with each other as follows: The Processing Module performs sensing from the environment. This module can process the necessary information and take action for the situation it detects. Information that can be obtained by learning may be required when performing the procedure. In this case, this module formulates a problem and sends it to the Learning Module. The learning module, on the other hand, creates an answer for the

problem using the generated information and sends it to the processing Module. The Processing Module also decides what data will be stored in the Training Data repository. It also calls the Learning Module to execute the learning algorithm to generate new knowledge periodically as needed or when the Training Data contains new samples. The learned information is stored in the "Generated Knowledge" base.

The Learning Module has four important components: Learning Algorithm, Training Data, Problem and Answer. The characteristics of the training data, problem and answer formats are designed depending on the learning algorithm used in the learning module. We planned to use the sci-kit learn [30] Python library in the development of the learning module. File-based databases (SqLite, JSON or XML) were preferred as the Knowledge Base and Training Set.

In the Learning Module, it is planned to use decision tree algorithms together with sequential pattern mining and deep learning, as well as other supervised learning methods. The knowledge obtained by learning is stored in the "Generated Knowledge" database. The learning agent will use this knowledge to notify the student of the code error and display the necessary steps for the student to correct the error. In addition to these, it also suggests possible corrections to the student. Students often find it difficult to understand the cause of errors, as the notifications from the Python interpreter are in English and not informative enough for beginners. Thanks to the proposed model, the student will be informed in Turkish and will be able to understand the reason for this error and will not waste time on trial and error to fix the code. Therefore, the learning process will be shortened and facilitated.

When the proposed Python code editor is first installed on the user's computer, the learning agent's Training dataset will contain data from some of the basic syntax errors that students often make. The learning agent training dataset will grow as the student uses the Python editor. In addition, if there is not enough data in the training dataset for the learning agent to make a decision, the training set can be updated over the Internet. The training dataset is planned to be kept in the cloud storage environment of the Code Storage module.

3.3. Python Interpreter Layer

The Python interpreter is available from the Python Software Foundation (PSF) website. As support for Python 2.X series interpreters may end in the near future, only Python 3.X series interpreters are planned to be supported in the proposed editor.

Cloud Based Code Storage Module consists of a web based code storage system and authorization system like "github". This module communicates with the code editor designed as a desktop application via web services. Thanks to this module, users will be able to store their projects written in the Python editor in the cloud and transfer them back to the editor when they need it. The Web Based Management Module gains functionality when the editor is used in schools and educational institutions. With this, teachers will be able to see the student's mistakes, observe the most common mistakes, get statistical information and reports for all students in the class, and also assign homework to students.

4. Results and Discussion

In this study, firstly, a detailed research on interactive guidance studies with intelligent agents is presented. Then, it introduces a concept for designing a dialog-based Turkish Python code editor that can analyze user errors. It is aimed to develop an open source Turkish supported integrated development environment (IDE) for the open source Python programming language. One of the most important innovative features of the study is that the code editor has an intelligent learning system that will analyze the errors of the users and provide guidance to the user. The design also has dialog based agent. The integrated development environment (IDE) outlined in this study is actually a complex system that brings together structures such as machine learning, intelligent agents, dialog-based agents, and chatbots. In this proposed system, users' coding styles, syntax errors and the loops and decision-making structures they use are analyzed by machine learning methods. According to the results obtained, users are guided by the dialog-based agent and the chatbot. In this way, users, especially beginners, can be given basic error-free coding practice without an instructor. The coding editor can be used as a stand-alone desktop software, or it will have the ability to store python source codes in the cloud. The ability to receive assignments assigned to the user by the trainer/teacher in the cloud environment and to send the students' solutions back to the teacher has also been designed. It will also enable error profile analysis of each user. In this way, it is planned to see to what extent the learning has been achieved and in which subjects the students have deficiencies.

Acknowledgement

This Project is supported by TUBITAK 1003 Prioritized Areas R&D Grant Program, Project No: 118E882

References

[1] Roesler, Marina, and Donald T. Hawkins. "Intelligent Agents: Software Servants for an Electronic Information World (And More!)." *Online* 18.4 (1994).

[2] Harmon, Paul. (Ed) (1995). Software agents. Intelligent software strategies, 11(1), 1-13. (January, 1995.)

[3] Russell, S. J. "Norvig (2003)." *Artificial intelligence: a modern approach* (2003): 25-26.

[4] Maes, Pattie. "Agents that reduce work and information overload." *Readings in Human–Computer Interaction*. 1995. 811-821.

[5] Kearsley, Greg. "Intelligent agents and instructional systems: Implications of a new paradigm." *Journal of Interactive Learning Research* 4.4 (1993): 295.

[6] Giraffa, Lucia Maria Martins, and Rosa Maria Viccari. "The use of agents techniques on intelligent tutoring systems." *Computer Science, 1998. SCCC'98. XVIII International Conference of the Chilean Society of*. IEEE, 1998.

[7] Giraffa, L. M. M., M. A. Nunes, and R. M. Viccari. "Multi-Ecological: an Intelligent Learning Environment using Multi-Agent architecture. MASTA'97: Multi-Agent System: Theory and Applications." Proceedings... Coimbra: DE-Universidade de Coimbra (1997).

[8] Chen, Weiqin, Roger Heggernes Pedersen, and Øystein Pettersen. "CoLeMo: A collaborative learning environment for UML modelling." *Interactive Learning Environments* 14.3 (2006): 233-249.

[9] Han, Keun-Woo, EunKyoung Lee, and YoungJun Lee. "The impact of a peer-learning agent based on pair programming in a programming course." IEEE Transactions on Education 53.2 (2010): 318-327.

[10] Gulz, Agneta, et al. "Building a social conversational pedagogical agent: Design challenges and methodological approaches." *Conversational agents and natural language interaction: Techniques and effective practices*. IGI Global, 2011. 128-155.

[11] Haake, Magnus, and Agneta Gulz. "A look at the roles of look & roles in embodied pedagogical agents–a user preference perspective." *International Journal of Artificial Intelligence in Education* 19.1 (2009): 39-71.

[12] Walker, Erin, Nikol Rummel, and Kenneth R. Koedinger. "Designing automated adaptive support to improve student helping behaviors in a peer tutoring activity." *International Journal of Computer-Supported Collaborative Learning* 6.2 (2011): 279-306.

[13] Tegos, Stergios, Stavros Demetriadis, and Anastasios Karakostas. "Promoting academically productive talk with conversational agent interventions in collaborative learning settings." *Computers & Education* 87 (2015): 309-325.

[14] Roos, S. (2018). Chatbots in education: A passing trend or a valuable pedagogical tool?. Msc Thesis, Uppsala University.

[15] (2018). Artificial Intelligence Markup Language (AIML). https://pandorabots.com/docs/aiml/aimlbasics.html. [Online; accessed 2018-05-15].

[16] Gehl, R. W. 2014. Teaching to the Turing Test with Cleverbot. Transformations: The Journal of Inclusive Scholarship and Pedagogy, 24(1-2): 56–66.

[17] Palasundram, K., Sharef, N. M., Nasharuddin, N., Kasmiran, K., & Azman, A. (2019). Sequence to sequence model performance for education chatbot. *International Journal of Emerging Technologies in Learning (iJET)*, *14*(24), 56-68.

[18] Goel, A., Creeden, B., Kumble, M., Salunke, S., Shetty, A., & Wiltgen, B. (2015, September). Using watson for enhancing human-computer co-creativity. In *2015 AAAI fall symposium series*.

[19] Lip ko, H. Meet Jill Watson: Georgia Tech's first AI teaching assistant; https://pe.gatech.edu/blog/meet-jill-watson-georgia-techs-first-ai-teaching-assistant, 5 Jan 2018, 05 Jan 2018.

[20] https://gonerdify.com/nerdybot (Access Date: 20.09.2021)

[21] Molnár, G., & Szüts, Z. (2018, September). The role of chatbots in formal education. In *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)* (pp. 000197-000202). IEEE.

[22] Singh, J., Joesph, M. H., & Jabbar, K. B. A. (2019, May). Rule-based chabot for student enquiries. In *Journal of Physics: Conference Series* (Vol. 1228, No. 1, p. 012060). IOP Publishing.

[23] Bird, Steven. "NLTK: the natural language toolkit." *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*. 2006.

[24] https://www.nltk.org/ (Access Date: 20.09.2021)

[25] https://github.com/microsoft/pyright (Access Date: 20.09.2021)

[26] http://flake8.pycqa.org/en/latest/ (Access Date: 20.09.2021)

[27] https://pypi.org/project/pyflakes/ (Access Date: 20.09.2021)

[28] http://pychecker.sourceforge.net/ (Access Date: 20.09.2021)

[29] Śnieżyński, Bartłomiej. "An architecture for learning agents." International Conference on Computational Science. Springer, Berlin, Heidelberg, 2008.

[30] https://scikit-learn.org/stable/ (Access Date: 20.09.2021)