Received: 15 December 2017 | Accepted: 5 May 2018
DOI: 10.1002/cae.21988

SPECIAL ISSUE ARTICLE

WILEY

SAIL—Software system for learning AI algorithms

Drazen Draskovic | **Milos Cvetanovic** | **Bosko Nikolic**

Department for Computer Engineering and Informatics, School of Electrical Engineering, University of Belgrade, Belgrade, Serbia

Correspondence
Drazen Draskovic, Department for Computer Engineering and Informatics, School of Electrical Engineering, University of Belgrade, Belgrade, Serbia.
Email: drazen.draskovic@etf.bg.ac.rs

Abstract

Artificial intelligence (AI) comprises a large spectrum of groups of algorithms: heuristic algorithms for search and planning, formal methods for representation of knowledge and reasoning, algorithms for machine learning and many more. Since these algorithms are complex, there is a need for a system which would enable their application both in everyday work and education processes. This paper describes a software system for learning AI algorithms called SAIL (Software System for AI Learning), which can be used both on computers and mobile devices. The paper gives examples of lab exercises and self-study tasks that through graphic representation and detailed procedures help students master this area. Students can enter their examples into the system and obtain correct solutions for those examples. At any point when an example is simulated, a student can proceed to the next step or go back to the previous one, save the current simulation as a file, or print the detailed procedure as a task solution. SAIL helps lecturers go through the syllabus more efficiently and improve class material, while at the same time it helps students get a better grasp of implemented algorithms. SAIL can also benefit software engineers, who can select and simulate an adequate algorithm to solve a specific problem. The results of the SAIL system are verified within the AI introductory course at the School of Electrical Engineering University of Belgrade and they are presented in this paper.

KEYWORDS

gamification in education, intelligent systems, knowledge representation and reasoning, mobile learning, search algorithms

1 | INTRODUCTION

Artificial intelligence (AI) is a scientific discipline which aim is to study and develop intelligent machine and software, that perform tasks requiring human intelligence [38]. Some examples of application of algorithms in this area include medical diagnosis, industrial system control and monitoring, finance and stockmarket transactions, gaming industry and other projects used in various fields [26]. Many software engineers and computer science students learn pseudo codes of these algorithms and generally can demonstrate how algorithms work on smaller problems. However, their understanding of the algorithm itself remains superficial, so when encountered with bigger problems, they cannot apply their knowledge. To understand algorithms in more depth, one needs an interactive environment, which would enable a step-by-step execution of each algorithm, observation, and detailed analysis of algorithm behavior from beginning to end. In addition, such an environment should enable user to see changes in the behavior of algorithms. Moreover, to learn and differentiate among algorithms, it would be good for the system to contain a simulation of parallel representation of various algorithms [8].

The popularity of software systems in education increased with the development of internet and mobile technologies [31].

Software systems developed at the School of Electrical Engineering University of Belgrade have been successfully used for several years [12,16,42]. It has been demonstrated that students understand problems and learn far better when they get answers to questions while following a dynamic graphic demonstration [19].

E-learning, distance learning, and self-learning techniques are applied to a great extent in the area of artificial intelligence [28]. Popular learning management systems (LMS), which are used for e-learning and distance learning, combined with specific software systems, have positive effects on student learning. This paper presents a software system for learning AI algorithms called SAIL (System for Artificial Intelligence Learning). The main goal of the development of this system is the implementation of algorithms and strategies from the area of artificial intelligence and their interactive simulation, which helps students grasp the content better, following the execution of algorithms and strategies through examples from the course material—a collection of tasks, or through their own problems which they enter into the system. This software system enables teachers to demonstrate the content through interactive simulation and animation, rather than only through slide presentations, as often is the case. SAIL can also be used by software engineers who can get an insight into possible effects of algorithms, which helps them select an adequate algorithm to fit specific needs, or use simulation to optimally set the parameters which they would use with the selected algorithm.

The following section presents reasons and motivation for introducing this software system, as well as a description of existing visual simulators for AI. The description of SAIL is given in Section 3. Section 4 presents examples of tasks and lab exercises for self-study using a mobile application within the SAIL tool, as well as a possibility to apply these algorithms through a process of gamification. Section 5 provides the analysis of results of application of the described software system in the process of education. The paper ends with a conclusion.

2 | PROBLEM DESCRIPTION AND RELATED WORK

When we redesigned the artificial intelligence course, we used the guidelines of IEEE Computer Society and ACM (Association for Computing Machinery), the organizations for the field of computing. According to these guidelines, a student should have an option to take at least one and maximum three courses on artificial intelligence and intelligent systems [10]. Learning outcomes should be: understanding basic concepts, ability to identify potential and situations when an intelligent system should be used, and ability to design, implement, and evaluate an intelligent system [11]. Study of artificial intelligence involves: heuristic algorithms for search and planning; formalisms for representation of knowledge and reasoning; machine learning techniques; methods applied to problems such as speech and language recognition, computer vision, and robotics.

Area of AI has been studied at the School of Electrical Engineering University of Belgrade since 1987 and course curriculum and learning outcomes are in accordance with the aforementioned guidelines [9]. Since 2003, when Serbia's higher education system was reformed, there have been two study programs where this area of study is included—Computer Engineering and Software Engineering. Currently, the AI course titled Intelligent systems is a mandatory course for fourth year undergraduate students of Computer Engineering and an elective course for students of Software Engineering, with the following weekly distribution of teaching hours: 2 classes for lectures, 2 classes for auditory exercises and problem solving, and 1 class for lab exercises. This course carries 6 ECTS (European Credit Transfer System). Lectures focus on the theoretical concepts and lessons. Auditory exercises consist of analysis and problem-solving from areas taught in lessons, while lab lessons aim at interactive demonstration using software system simulation and are necessary for a better understanding of complex topics. Table 1 gives an overview of topics and subtopics, level of coverage on the course, as well as electiveness suggested by the guidelines. Other more advanced topics are covered by other courses in later years of graduate and master studies. These include: Neural Networks, Data Mining, Machine Learning, Robotics and Automation, Robotics Systems Theory, Pattern Recognition, Natural Language Processing, etc.

Table 1 shows that all four mandatory topics recommended by the guidelines are completely covered, with three more topics covered to a smaller or a larger extent. Representation of knowledge and algorithms covered by the Intelligent systems course are thematically very diverse and comprise: search algorithms—uninformed search (breadth-first, depth first, depth-first with iterative deepening) and heuristics and informed search (hill-climbing, best-first, branch and bound, A*), Minimax search and Minimax with alpha-beta pruning, resolution theorem proving, fuzzy logic, semantic networks and frames, production systems, representations for reasoning under uncertainty, planning systems, inductive systems, and Bayesian networks. The listed algorithms done in lectures and auditory exercises can in some cases seem quite abstract and unclear to students, so interactive simulations of those algorithms facilitate their understanding and learning [13].

Upon analyzing related work, recommendations, experiences in development of educational tools and teaching of AI algorithms, it was concluded that the simulator should satisfy the following requirements:

R1) common graphic user interface (GUI) for all simulations, intuitive commands and toolbar options, which make the system user-friendly;
R2) a high level of interaction between the user and the system;

TABLE 1 Topics covered in the Intelligent systems course

Topics [M]andatory/[E]lective	Level of coverage	Covered subtopics
Fundamental issues [M]	⊕	Overview of AI problems; Problem characteristics; Nature of agents
Basic search strategies [M]	⊕	Problem spaces, problem solving by search; Factored representation; Uninformed search; Heuristics and informed search; Space and time efficiency of search; Two-player games; Constraint satisfaction
Basic knowledge representation and reasoning [M]	⊕	Predicate logic; Resolution and theorem proving; Forward chaining and backward chaining; Review of probabilistic reasoning, Bayes theorem
Basic machine learning [M]	⊕	Inductive learning; Simple statistical-based learning, decision trees
Advanced search [E]	⊕	Constructing search trees, dynamic search space; Stochastic search—simulated annealing, genetic algorithms; A* search, beam search; Minimax search, alpha-beta pruning
Advanced representation and reasoning [E]	○	Semantic networks
Reasoning under uncertainty [E]	●	Knowledge representation—Bayesian Networks; Decision Theory

○, Partial level of topic coverage; ●, Substantial level of topic coverage; ⊕, Full level of topic coverage.

R3) step-by-step simulation for each algorithm and real-time monitoring of simulation/algorithm;
R4) possibility of comparative analysis between algorithms;
R5) loading predefined problems and executing within the tool;
R6) possibility to make own examples, change parameters, save the current example and load earlier examples;
R7) consistency of representations and descriptions of implemented algorithms and consistency of user interface;
R8) ability to use the help menu in each step of the execution of the algorithm;
R9) a multiplatform system involving both desktop and mobile applications and self-study using mobile devices; mobile technologies are used more and more for educational purposes, but publicly available mobile applications for studying intelligent systems still do not exist and are not used in coursework.

The listed requirements served as criteria for analysis of software systems used as tools in artificial intelligence courses at universities around the world. The rest of this chapter analyzes the most important examples that can be found in open literature, identifying their advantages and disadvantages.

"Chinese room" [43] simulator is used at Illinois University in the artificial intelligence course. This topic is usually introduced at the beginning of the course, when the concept of computer modeling of human thinking is discussed. The disadvantage of this simulator is the fact that everything is done through animation and that there is no opportunity for interaction with the user.

"MIT Open Course Ware" simulator [35] is a tool used in artificial intelligence courses used at Massachusetts Institute of Technology and Chicago University. It consists of six units: planning, search, constraint satisfaction, biological mimetics, other forms of learning, and Bayesian networks. Planning is done using the STRIPS (STanford Research Institute Problem Solver) algorithm, and the supported search algorithms are depth-first and breadth-first search algorithms, hill-climbing, best-first, branch and bound, and A* algorithm. As far as other algorithms and strategies are concerned, this system supports a minimax algorithm and an alpha-beta cut-off algorithm, constraint satisfaction method, realized as a map coloring problem, genetic algorithms, self-organizing neural networks, and Bayesian networks. The main setback of this simulation is that users are neither able to enter their own problems for search nor observe how algorithms are executed step by step.

"AI Search" simulator [2], developed at RMIT University in Australia, demonstrates how various search algorithms work on the example of "8 Puzzle" game, which is used in lab exercise classes in artificial intelligence. Evaluation shows that students who were in the experimental group and used this simulator understood the content far better and had better results in comparison to the control group, who did not use this simulator [32]. The simulator is very interactive and contains a detailed description of execution of a selected algorithm. The main limitation of this simulator is that this game shows only search algorithms.

Software system "CIspace"/"AIspace" [4] was developed at the University of British Columbia. It represents a collection of interactive tools and covers eight areas: graph search, solving the constraint satisfaction problem based on consistency and stochastic local search, inference, Bayesian networks, inference trees, neural networks, and conversion of STRIPS algorithms into CSP. The system supports interactive simulations, which gives the user partial control over examples [5]. This software system is a good learning support tool for this field, but it does not cover all compulsory topics from AI area. It is not possible to compare two different algorithms.

"AIMA3e" simulator [3] is based on all algorithms from the most famous book in this area [38]. This system contains 7 simulations and 13 textual demos, which demonstrate the work of particular algorithms of artificial intelligence. It gives a detailed overview of search algorithms, popular games from the gaming theory field, and games based on the constraint satisfaction problem (CSP). Similar to the previous system, this system does not cover all suggested AI topics either, and the user is unable to influence the examples given as demo animations.

"Searcher" simulator [39] is a helpful teaching and learning tool for educators and students dealing with algorithms. It is used at Liverpool University and it is very convenient because its good graphic representation makes it easy to follow animations, which is its greatest advantage. The disadvantage is that it is based only on search algorithms.

"Simple Expert System" [40] represents an overview of a simple expert system which gives conclusions based on a knowledge base. This simulator enables the user to select one of the offered knowledge bases, or create their own, complying with the format rules for production and facts. The user answers the questions with true or false, while the system uses these answers to draw an inference using productions and facts. The setback of this simulation is the fact that it cannot save a knowledge base or load a knowledge base from file, and that it is based only on inference systems.

"Sudoku Solver" simulator [41] is a tool which uses constraint satisfaction methods to demonstrate Sudoku problem solving. The user can load an example from the library, independently create a new Sudoku example, enter values into table cells, and solve Sudoku without assistance. When the user enters a value which does not comply with some of the constraints, the field with an incorrect value will be clearly marked. The disadvantage of this simulator is that it uses the game of Sudoku only to demonstrate algorithms from the area of CSP.

"AI Exploratorium" system [1] demonstrates the mechanisms of the decision tree (decision tree learning) and IDA* search algorithms (iterative deepening depth-first search algorithm). Its disadvantage is that the realized algorithms show only the basis of machine learning, without enabling the user to create and execute their own examples.

Software systems based on artificial intelligence are often applied in simulation of sports games, war games or lottery [7,24,36]. Although such systems realize a large number of various algorithms in the backend part, it is harder to explain how algorithms work, and they are not convenient for self-study.

"Multi-Agent Soccer" simulator [7] is based on a simplified model of football and can be used in introductory coursework on artificial intelligence. The advantage of this simulator is that it covers a large number of algorithms and that users can enter their own data. The main disadvantage, however, is the fact that it is not possible to examine individual algorithms and their step-by-step execution.

A similar system, called "RoborSoccer" [24], implemented search algorithms and genetic algorithms. The system is visually very clear but its disadvantages are its weak interaction with the user and the lack of option to review individual steps of algorithms.

Equally important is the MLeXAI project (Machine Learning Experiences in AI) [37], developed for artificial intelligence learning, focusing on machine learning application. The aim of the project is to develop an adaptable system for computer presentation of artificial intelligence topics in order to show the close connection of computer science and artificial intelligence [30]. This complex system is designed for use in the form of six hands-on lab projects: classification of web documents, which is used for learning the basics of data mining and key algorithms of machine learning; web user profiling; neural network character recognition; solving the N-Puzzle problem using search algorithms; solving the Dice Game Pig problem with core concepts of reinforcement learning; and Clue Deduction for knowledge representation and reasoning. A good side of this project is its diversity of topics and an easy integration of tools into an introductory AI course. On the other hand, the downside is the fact that there is no uniformity and that the tools comprising the MLeXAI system are diverse. In some lab tasks, MLeXAI also relies on results obtained from other systems, such as WEKA.

The WEKA system is a collection of algorithms from the area of machine learning [21,44]. It is possible to apply these algorithms directly to the input data collection, and it is possible to retrieve an algorithm from a code written in the Java programming language. WEKA offers a data processing option involving: preprocessing, classification, regression, clusterization, and visualization. Some of the supported algorithms for decision tree construction are: ADTree, BFTree, and ID3. This system is quite complex, and gives good visualization and explanations to the user. The main disadvantage of this system is that its work focuses on data mining and machine learning subareas.

Table 2 gives an overview of topics and the degree to which these topics are included into the analyzed systems of artificial intelligence learning. The analyzed software systems designed for specific courses at various international universities are successfully applied in coursework.

Table 3 gives basic information on software systems—the university where the system is implemented and its programming language, followed by an evaluation of software systems based on their main characteristics (requirements R1-R9). Visual representation is a graphical representation of the algorithm and it is expressed at three levels: excellent, good, and bad. Controlling the pace of execution, an algorithm is marked with SS if the "step by step" is supported, and with R if the simulation is running at a constant speed until it reaches the solution. For other traits, the presence of certain features is marked with "yes" and the absence with "no."

From our analysis of the described software systems used for AI learning we can conclude that the most comprehensive systems are "MIT Open Course Ware," "AIspace," and "AIMA3e." They have a very good user interface and cover a larger number of core AI topics, which makes them the most popular tools in AI coursework. The analysis has also shown that no system entirely meets user requirements (R1–R9). These are the main reasons why the authors of this paper developed their own software system—SAIL, which will be described in the following section.

3 | SAIL FEATURES

The educational software system SAIL consists of a desktop application written in the Java programming language and an Android mobile application. SAIL comprises 30 algorithms, which are grouped within 10 units, as presented in Table 4. Each unit contains algorithms and strategies which cover specific subtopics and has a common user interface so that users do not have to adapt to different environments.

Key SAIL characteristics are the following: data and example input by the user, system check of the entered data, step-by-step simulation of specific algorithm, possibility to go straight to the beginning or to the end of a simulation, loading examples from the predefined textual files and saving examples as textual files. Each step of the simulation is described in detail and each following system state is highlighted.

TABLE 2 Coverage of main topics in the area of artificial intelligence for analyzed systems

System	Basic concepts	Basic search strategies	Reasoning based on knowledge	Advanced search strategies	Advanced reasoning techniques	Machine learning	Planning
Chinese room	⊕	x	x	x	x	x	x
MIT OCW	●	⊕	⊕	⊕	○	●	●
AI search	●	⊕	x	x	x	x	x
CIspace/AIspace	●	⊕	⊕	⊕	○	●	●
AIMA3e	●	⊕	⊕	⊕	○	●	x
Searchr	●	⊕	x	●	x	x	x
Simple ES	●	x	●	x	x	x	x
Sudoku solver	●	○	x	x	x	x	x
Minesweeper	●	○	x	x	x	x	x
AI exploratorium	●	○	x	x	x	○	x
Multi-agent soccer	○	⊕	●	○	○	x	x
RobotSoccer	●	○	x	⊕	x	x	x
MLeXAI	●	●	●	●	○	⊕	x
WEKA	●	x	○	x	○	⊕	x

○, Partial level of topic coverage; ●, Substantial level of topic coverage; ⊕, Full level of topic coverage; x, Topic not covered.

TABLE 3 General information of the analyzed software systems and evaluation based on main characteristics

System name	University	Program language	R1	R2	R3	R4	R5	R6	R7	R8	R9
Chinese room	State Illinois University, US	Flash	Excellent	No	R	No	Yes	No	No	Yes	No
MIT open course ware	Massachusetts Institute of Technology, US	Java	Excellent	Yes	R	No	Yes	No	Yes	Yes	No
AI search	RMIT University, Melbourne, Australia	Java	Excellent	Yes	SS + R	Part	Yes	Yes	No	Yes	No
CIspace/AIspace	The University of British Columbia, Canada	Java	Excellent	Yes	SS + R	Part	Yes	Part	Yes	Yes	No
AIMA3e	by several universities and researchers [3]	Java	Excellent	Yes	SS + R	No	Yes	No	Yes	No	No
Searchr	University of Liverpool, UK	Coffee-Script	Excellent	Yes	SS + R	No	Yes	Yes	No	Yes	No
Simple expert system	University of Kent, UK	Java	Good	Yes	R	No	Yes	Yes	No	No	No
Sudoku solver	University of Nebraska, US	Java	Excellent	Yes	SS + R	No	Yes	Yes	No	Yes	No
Minesweeper	University of Nebraska, US	Java	Excellent	Yes	SS + R	No	Yes	Yes	No	Yes	No
AI exploratorium	University of Alberta, Canada	Java	Excellent	Yes	SS + R	No	Yes	No	No	Yes	No
Multi-agent soccer	Delft University of Technology, Netherlands	Java	Excellent	Yes	R	No	Yes	Yes	Yes	Yes	No
RobotSoccer	Adelaide University, Australia	Java	Good	No	R	No	No	Yes	No	No	No
MLeXAI	University of Hartford, US	Java/Lisp/Prolog/ Matlab	—	No	R	No	Yes	No	No	No	No
WEKA	University of Waikato, New Zealand	Java	Excellent	Yes	R (with details)	Part	Yes	Yes	No	Yes	No

R1, visual representation; R2, interactive simulation; R3, flow control of algorithm execution; R4, comparison of several algorithms; R5, predefined problems; R6, new problems making; R7, consistency; R8, help; R9, mobile app; SS/R, step by step tempo/running tempo; part—partially.

TABLE 4 Weekly schedule of Intelligent system course

Schedule	Units	Subject of unit	#Num	L
W 1		Introduction in AI. Uninformed search	1	
W 2	U 1	Heuristics and informed search	7	+
W 3		Introduction in game theory		
W 4	U 2	Algorithms of game theory	2	+
W 5, W 6	U 3	Formal logic	4	+
W 7	U 4	Semantic networks and frames	2	
W 8	ME	Midterm exam that includes first four units.	/	
W 9	U 5	Production and analytical systems	4	+
W 10	U 6	Reasoning under uncertainty	2	
W 11	U 7	Fuzzy logic	1	
W 12	U 8	Problem solving strategies	5	+
W 13	U 9	Bayesian networks	1	
W 14, W 15	U 10	Basics of machine learning Induction algorithm. Decision trees.	1	+

W, week; U, unit; ME, midterm exam; #Num, number of simulations include in the SAIL; L, have lab exercise.

The part that follows describes the parts of SAIL, the computer desktop application, and the mobile device application.

3.1 | Desktop application

The first step for the student is to select an algorithm to execute in the system. SAIL includes search algorithms, game theory algorithms, first-order formal logic, production systems, semantic networks and frames, problem solving algorithms GPS and STRIPS, undetermined environment algorithms, constraint satisfaction algorithms, ID3 decision tree induction algorithm and Bayesian networks. All these algorithms can be divided into three groups: algorithms whose solutions are only textually described, algorithms whose solutions are represented as graphs and/or trees with a textual description, and algorithms which have a specific visual description in frontend, such as X-O games, moving blocks puzzles, crosswords, Sudoku, etc. Figure 1 shows the introductory screen of computer version of SAIL.

When a specific algorithm is selected, one can load the previously saved data or enter new data. Data are saved in

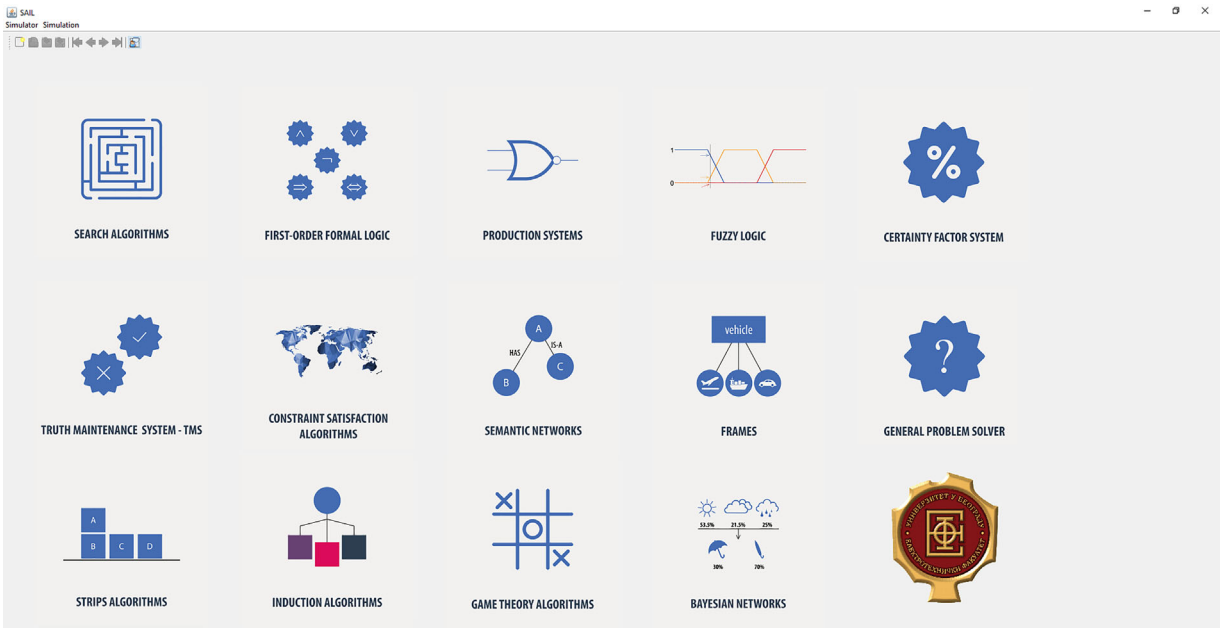


FIGURE 1 The introductory screen of the SAIL

text files with a specific extension depending on the simulation.

The entry of simulation of parameters is most often carried out in a separate window at the very beginning of algorithm work. The main window is split into several units and serves to monitor relevant simulation states and output data writing, whereby students can observe each step of the algorithm, that is, the strategy that they are learning. In each step students can see visually and textually how the state has changed, so that at the end of the simulation they can understand what the aim of the requested algorithm is, and how it works. If students do not have theoretical knowledge of an algorithm, and they do not know which data are followed throughout its execution, they can learn this by using the help menu of the system.

The majority of realized algorithms presents learning using graphs and trees. Search algorithms included in the SAIL aim at finding a path from the initial to the target node. In order to give a better overview of a simulation, the system shows a search tree. Certain search algorithms use a heuristic function, while SAIL also uses the principle of dynamic programming, which in a combination with realized algorithms most often executes an algorithm in fewer steps and gives better search performance.

The other algorithms like GPS (General Problem Solver) and STRIPS are based on a textual presentation of solution. They aim at demonstrating how to create a system able to solve problems, so they are introduced at the beginning of the course. Apart from the textual display of algorithm execution, STRIPS also shows a block animation, with the aim of bringing the blocks from the initial state to the target state using permissible operators. Each step of these implemented algorithms presents the textual representation of algorithm states, so students can easily learn and understand their work.

Graph drawing is very simple and intuitive. It is necessary to draw nodes, connect them with links, and mark the initial and the target node. There are three modes for graph drawing: graph editing, nodes picking, and graph transformation. When a graph is edited, one can change the name of the node and assign it a heuristic function by clicking on the node. When a node is removed, all links between that node and other graph nodes are also erased. A link is created by clicking on a node and drawing the cursor to the other node. Each link can be assigned path cost, and one can also choose its directionality. The other work mode is node picking, when a node is selected and then moved with the aim of getting a better visual outline of the graph. The visual graph presentation option can also be rearranged automatically.

When it comes to gaming theory, each turn-based game consists of problem space, initial state, and target state. The nodes of the tree are positions in the game, while branches represent possible moves which shift one position into another one. The root of the tree signifies the initial state, while the leaves signify terminal states—victory, tie, or defeat. The games realized in this learning system are less complex—Tic-Tac-Toe (Figure 2) and Nine Men's Morris, so that it is easier for students to understand the minimax algorithm and minimax algorithm with alpha-beta cutoff.

SAIL implements four algorithms used in first-order logical inference: conjunctive normal form (CNF) algorithm and three algorithms for resolution proving. These algorithms

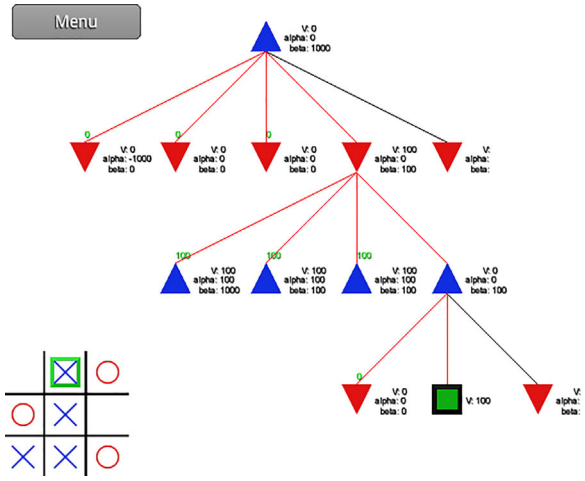


FIGURE 2 The game theory algorithms in the mobile version of SAIL

belong to the group of algorithms where the textual display is very important especially due to special symbols of mathematical logic. The first step in this simulation is to transform a formula into a CNF form. Students can use the following symbols: $AND, OR, NOT, (,), \Rightarrow, \forall, \exists$. When the clausal form is reached, the obtained facts become input data for a specific resolution method algorithm which can be executed in a following simulation: set-of-support strategy, unit preference strategy, breadth-first strategy.

Simulations for informal knowledge representation, semantic networks and frames, use a graphic simulation similar to search algorithms. When students define a semantic network, they enter objects, which consist of nodes in the shape of a circle and a rectangle, and links, which are directed lines with arrows (Figure 3). A node represents a specific class example called instance. There are two links: IS_A, which is used to mark element class, and PART_OF, which is used to mark that an element is a part of another element. In contrast to semantic networks, in frames each node has its own attributes and attribute values, so when a particular node is selected, all its attributes and values are shown.

Production systems are the most frequent form of representation of knowledge in intelligent systems due to their cause-effect structure. When defining a production system, students need to define the content of working memory and production memory, and then select an interpreter which carries

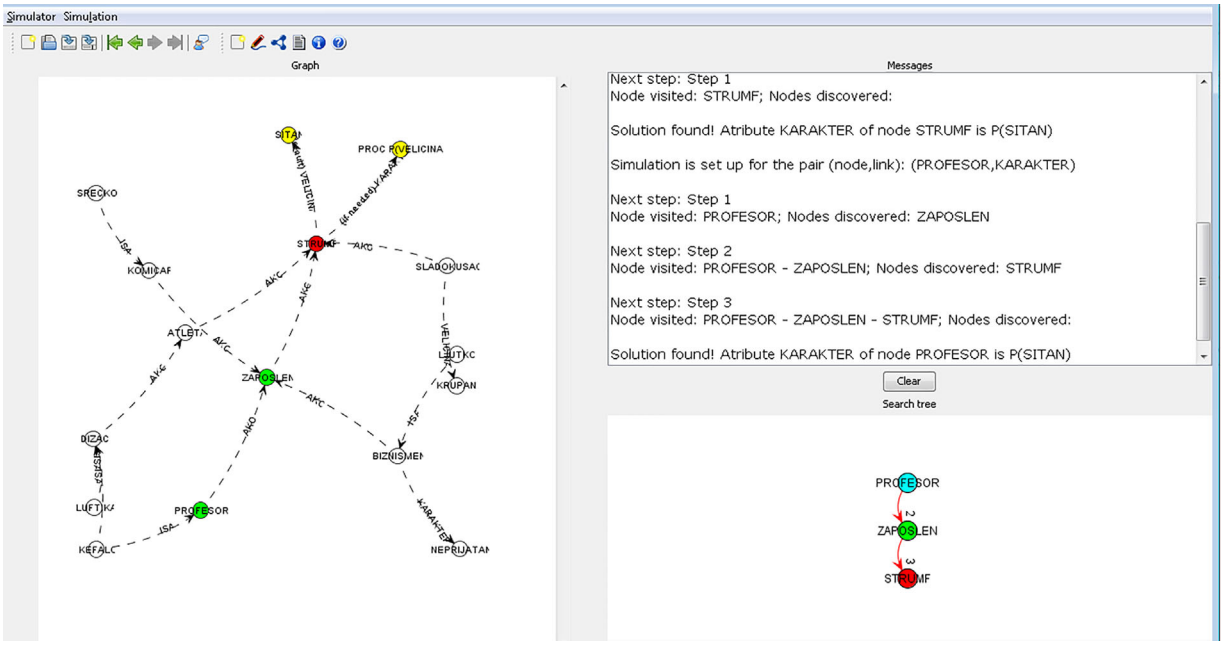


FIGURE 3 Simulation which represents informal knowledge

out the given inference algorithm. In SAIL, facts are presented as an ordered triple consisting of indicators, attributes, and values which describe a certain quality. When students enter the initial simulation content, they need to select one of the inference algorithms: direct chaining algorithms (data-driven), backward chaining (goal-driven), hybrid chaining, and Rete algorithm. During a simulation, it can happen that all conditional elements of a rule are met, which is when algorithms of prioritization are used to resolve the conflict within the given set. The available algorithms are: priority resolution, complexity resolution, resolution of the specific conflict, loading order resolution, and the triggering time resolution. Each of these algorithms provides a different sequence of simulation execution and influences the final result.

Work in an undetermined environment is shown in SAIL by means of three simulations: certainty factor, truth maintenance system (TMS), and fuzzy logic.

The certainty factor model contains a lot of formulas and calculations which used be quite difficult to understand without SAIL. In order to get a result and interactive steps of this model's execution, students must define the rules, observations, and conclusions using textual fields. Certainty factor (CF) quantifies the degree of trust in a certain inference and has a value within the scope $-1 \leq \mathrm{CF} \leq 1$. Rules have their predefined grammar and when they are defined, the reserved words IF, AND, OR, are used. Each observation also has its own certainty factor. When starting a simulation, in case there are multiple inferences, the user can select the one they wish to consider. The simulation is carried out step by step and at any point the student can see the formulas which are used and the calculation which is obtained.

TMS is a mechanism of saving information on dependencies and inconsistency recognition. Before a simulation starts, it is necessary to define the basis of knowledge whose consistency will be maintained by TMS. When the user defines all the rules in the system, they can initiate the change of state of a certain rule. When there is a change of rule, the TMS algorithm activates and maintains the consistency of the base. Each step of this algorithm is visible to the user. While a simulation is under way, no changes are allowed before it ends. When a simulation ends, the user can redefine or erase an existing rule and restart the algorithm.

Fuzzy logic user interface consists of text surface for entering and adapting the values of input variables, text surfaces for writing status messages, and three surfaces for graph display. Entering data are executed by loading the database in the format of the Fuzzy Control Language (FCL) or generated by FCL database through the wizard concept. A student should define the input and output variables, sets of these variables and the function of their membership to the set, algorithms used for aggregation, activation and accumulation, and rules. The graphs represent the function of set membership of output variables and the accumulated functions of membership. The following functions of membership are supported in SAIL: singleton, triangular, trapezium, and shoulder. The process of obtaining a crisp value is called concentration or defuzzification. There are several methods of defuzzification: center of gravity (COG), center of gravity if the output variables are discrete (COGS), center of area (COA), right of maximum (RM) and left of maximum (LM). Figure 4 shows the example of fuzzy inference system that calculate tip (output) based on service and food (input variables).

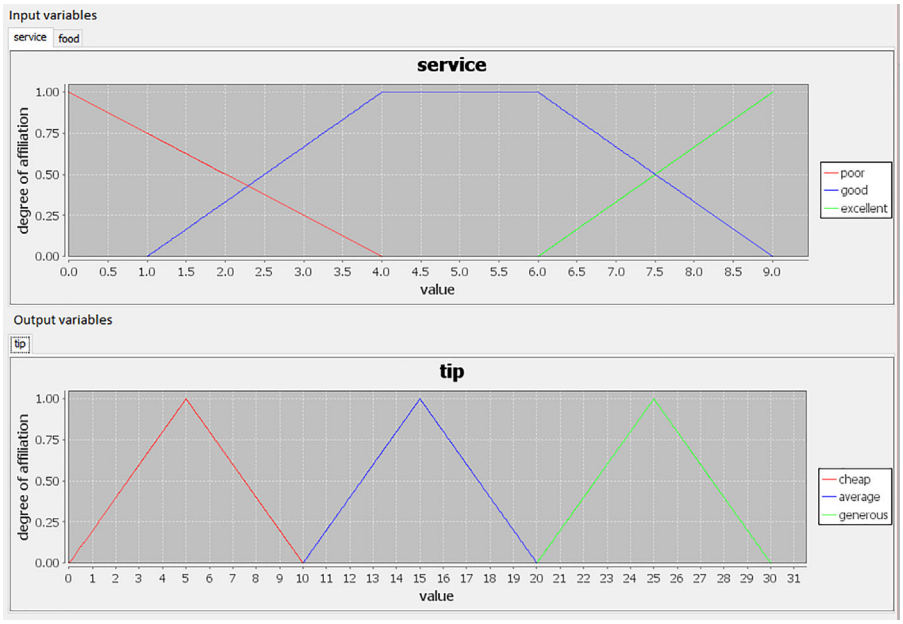


FIGURE 4 Fuzzy logic simulation

Constraint Satisfaction Problem (CSP) consists of variable domains of value which variables can have and which they have to fulfill. When solving these problems in our system, we apply the backtracking search, which is a type of depth-first search. Two ways to improve search implemented in SAIL are forward checking and arc consistency. In the SAIL system desktop application, the improved CSP algorithm was implemented using three problems—map coloring, N-queens problem, and Sudoku, while a crossword puzzle problem was implemented in the mobile application.

SAIL enables the formation of the Bayesian networks and demonstrates the inference process and decision making within that network. For each node of the graph we must determine its domain, that is, a collection of possible states where one can by chance find the variable that the node represents. Possible states must be mutually exclusive and cover all possible values which may contain a variable. After the nodes are introduced, one should form links. If one node causes another one, the arrow represents the direction of dependence. When a network is formed, students need to quantify the links between the connected nodes by specifying the causal probabilities for each link between a pair of nodes. This is how tables of caused probabilities are formed. After the network is formed and the tables of probability for each node are defined, one moves on to inference.

The induction algorithm that is implemented is the ID3 algorithm. The graphic representation of algorithm performance is realized in the form of a decision-making tree or textual representation of all calculated values. In SAIL, the implemented algorithm has the graphic form of the decision-making tree after each algorithm step is executed, using the step-by-step option. There are two ways in which students can enter data: by creating a table with input data directly from the system using control buttons, and entering data from a textual data file.

SAIL is realized in the programming language Java, which is supported by a large number of various platforms. In addition, the development of the system is made easier by using the existing open-source libraries for drawing graphs [20,27], which are despite certain downsides and limitations still a better solution compared to the independent development of a completely new package for graph visualization.

The basic property of the system architecture is modularity, so that the system can be efficiently upgraded by adding new simulations and algorithms. Section 5 explains how this system was developed modularly.

3.2 | Mobile application

In addition to the desktop version, an application for Android platform based mobile devices is also developed as an integral part of the SAIL. This mobile application enables students to learn independently using their mobile phones and tablets.

Simulations of all algorithms and strategies are realized in the similar way as they are in the programming language Java. When an application is started, students should select a topic and start a simulation. When the simulation of a certain algorithm starts, there is a blank area for simulation and four control buttons which respectively signify: jump back to the beginning of the simulation, take a step back, take a step forward, and jump to the end of the simulation.

In contrast to the desktop application, there was no satisfactory library for drawing graphs and trees for the Android version so it was necessary to develop a custom method of drawing. The tree is scaled in the way that the whole tree can fit the size of the screen but if needed, one can also zoom using the "pinch to zoom" option. By touching the forward button the user goes to a tree node, that is, the initial node of the entered graph. When a node is visited, at that point newly discovered nodes also enter the search tree. The currently processed node is marked in red color, the visited nodes are green, whereas the unvisited nodes are gray. The algorithm goes step by step until it finds the given node or as long as there are unvisited nodes.

There are two work modes: for entering graphs and for changing graphs. They are presented in the action bar at the top of the screen. At the bottom of the screen there is a button for selecting algorithm search (spinner) and a button for simulation initialization. The application is initialized in the full screen mode, without an action bar or a status bar, so that it is as large as possible for the user, which is very important for mobile devices with small screens. By touching the screen the user creates a new node. The node initially gets a new name from the set of available names, and has a value 0 for heuristics. If the user tries to add a node that would overlap with another already existing node, the newly created node takes that position while all other nodes with which it overlaps are moved away from it. By swiping from one node to another existing node, a link with 0 cost is created. If a student wants to change some of the properties of nodes or links, this is done in the graph changing mode by clicking on the button for changing in the top right corner of the screen. By tapping on the node, one opens a pop-up window where changes regarding features such as name, heuristics, and whether the node is initial or final can be made. A simulation cannot start running unless the initial and the final nodes have been marked. A node can be swiped to another position on the screen. This is when it is checked whether there is an overlap and other overlapping nodes are removed. When nodes are moved, the same happens with the links between them. A link can be updated or erased in the same way as a node. If a given algorithm uses search metrics, that metrics is presented above the nodes. If the heuristic function and the path cost is used, both values are added toward the final sum, as presented in Figure 5.

Figure 6 demonstrates a CSP on the example of a realized Crossword Solver model. The user needs to define table dimensions, the number of rows and columns, define the set

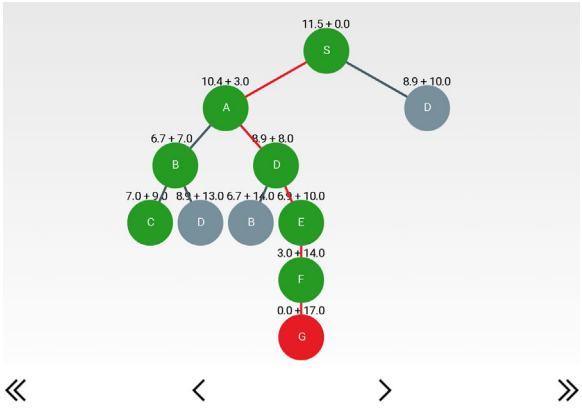


FIGURE 5 Tree for A* search simulation in mobile application in SAIL

of words, and when one initializes the algorithm, one can observe how it works step-by-step, with detailed explanations written below the table.

4 | LEARNING BASED ON SAIL

The Intelligent systems course consists of 10 units. It lasts 15 weeks, out of which 14 are for lessons and one is reserved for a mid-term test. During the semester there are also 6 lab exercises, which are done after certain topics are covered in lectures. For each topic there is a lab exercise done in SAIL. Students are not given grades for these exercises, but this is rather demonstration and practice designed with the aim of more effective learning. The course schedule and lab time slots are presented in Table 4.

In the rest of this section, an example of a lab exercise demonstrating the application of first-order logic algorithms will be presented. That will be followed by an example of a search algorithm for self-study on an Android mobile device. Finally, the possibility to upgrade the SAIL into a computer game which can serve for learning the basic concepts and algorithms from the AI field.

4.1 | Lab exercise

The following subsection shows the task "Family Ties" used in lab exercises for learning resolution algorithms (Table 4, Week 6, Unit 3). Students get the textual task with the following statements:

1. If person X is the brother of person Z and person Y is also the brother of person Z, then person X is the brother of person Y, or X and Y are the same person.
2. If person X is male and has the same mother as person Y, then X is the brother of Y or X and Y are the same person.
3. Marija is the mother of Milan and Ana.
4. Milan is male.
5. Jovan is Ana's brother.
6. Milan and Jovan are not the same person.
7. Milan and Ana are not the same person.

The task is to use the resolution strategy to either confirm or dispute the statement that Milan is Jovan's brother.

After a resolution algorithm is initialized, students enter facts into SAIL. It is possible to enter OR, negation, open or

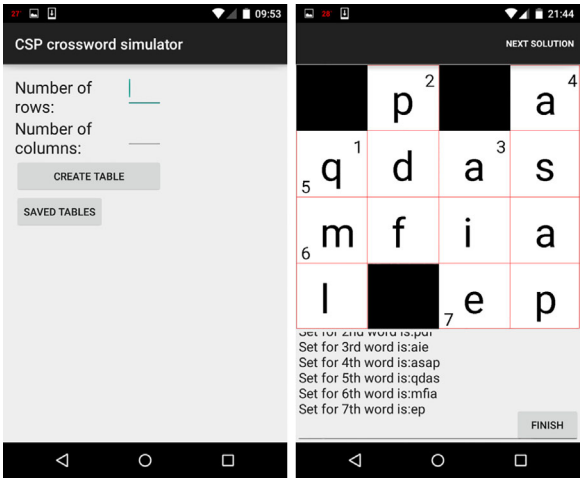


FIGURE 6 Android example Crossword Solver in SAIL

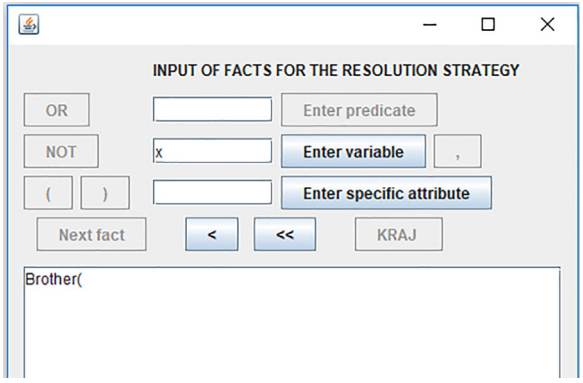


FIGURE 7 An example of the input of facts in the simulation of formal first-order logic

closed parentheses, predicate, variable or specific attribute, as presented in Figure 7.

In the lab exercise task, the student must first present the facts using predicative formulas in the following way:

1) $\forall x \forall y \forall z [Brother(x,z) \wedge Brother(y,z) \Rightarrow$
$\Rightarrow Brother(x,y) \vee Same\_person(x,y)]$

2) $\forall x \forall y \forall z [Male(x) \wedge Mother(z,x) \wedge Mother(z,y) \Rightarrow$
$\Rightarrow Brother(x,y) \vee Same\_person(x,y)]$

3) $Mother(Marija,Milan) \wedge Mother(Marija,Ana)$
4) $Male(Milan)$
5) $Brother(Jovan,Ana)$
6) $\neg Same\_person(Milan,Jovan)$
7) $\neg Same\_person(Milan,Ana)$

All statements apart from 1) and 2) already have a clausal form, since they are lateral. Statements 1) and 2) are brought into a clausal form by removing the implication and applying De Morgan's laws while renaming the variables of the other formula to ensure uniformity. To the previous statements the negation that Milan is Jovan's brother is also added, in order to look for contradiction applying the rule of resolution. Figure 8 shows the facts after they are entered into the system. The left part of the window shows all facts, while the right part of the window shows only the newly acquired facts, i.e. statements which are coupled, with appropriate unification. The lower part of the window shows explanations.

This example demonstrates the resolution application, with the breadth-first selection. First, all possible combinations of existing facts are considered, and after that the newly acquired facts are paired up. Figure 9 shows the final solution that students reach after completing ten simulation steps. One new fact is acquired in each step and added to the list in the right window of the simulation screen.

Upon completing an algorithm simulation, students can generate a report with a detailed solution description in the

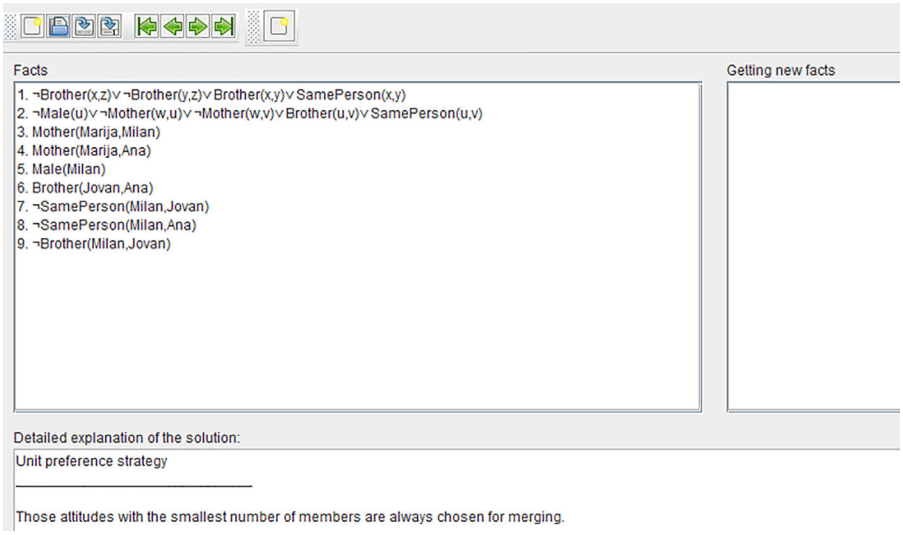


FIGURE 8 User interface, after entering the facts, in the simulation of the resolution algorithms

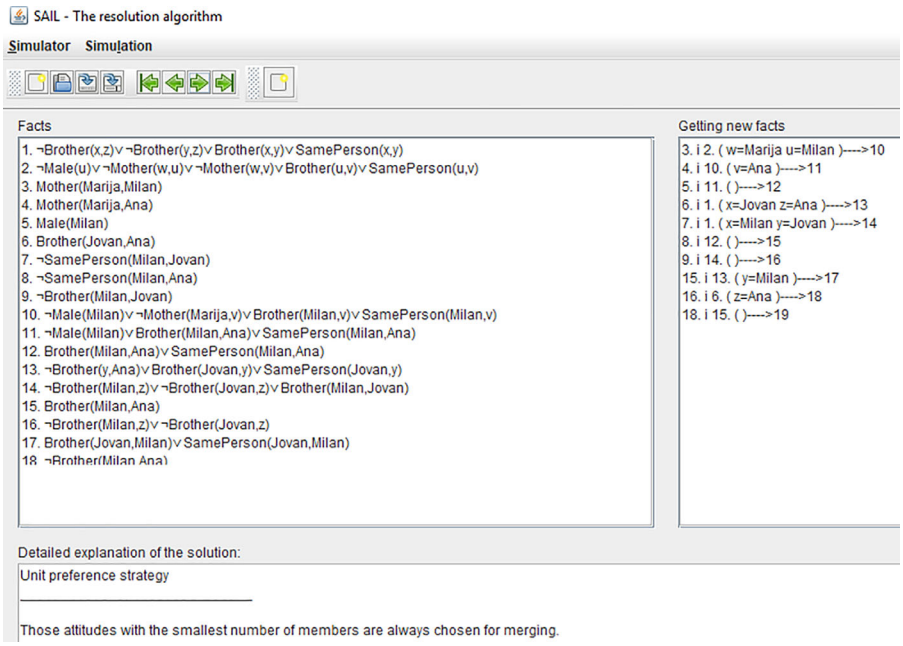


FIGURE 9 Example of the user screen after applying the resolution algorithm and breadth-first strategy

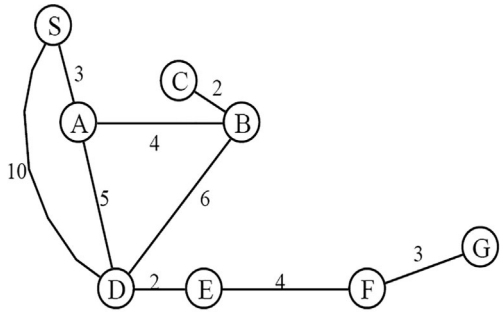
form of an external PDF file, which makes it possible to analyze the solution after the lab exercise task is completed. The students who do not complete the lab exercise, complete the task at the end of the exercise by observing a simulation run by the teaching assistant and completing those steps on their own computer.

4.2 | Mobile-assisted exercise for students' self-learning

Mobile learning (m-learning) represents a completely new form of distance learning, self-learning, and new expansion of e-learning [29]. Learning of AI content on smart phones using SAIL can be demonstrated on the example of search algorithms. One of the well-known problems of these algorithms is finding a path between cities for which one knows the path cost (distance between cities) and the heuristic function (HF). The student can draw a new graph or load an existing graph from a textual file.

When a graph is taken from a file, it is necessary that the file has the following structure so that the parser can load the right file:

- Initial node in graph: A 10.4 start
- Graph node: D 8.9
- Targeted graph node: G 0 end
- Link: A-D-5



City	S	A	B	C	D	E	F
Distance to city G	11.5	10.4	6.7	7.0	8.9	6.9	3.0

FIGURE 10 Graph and values of nodes heuristic function for path-finding in one self-learning lesson

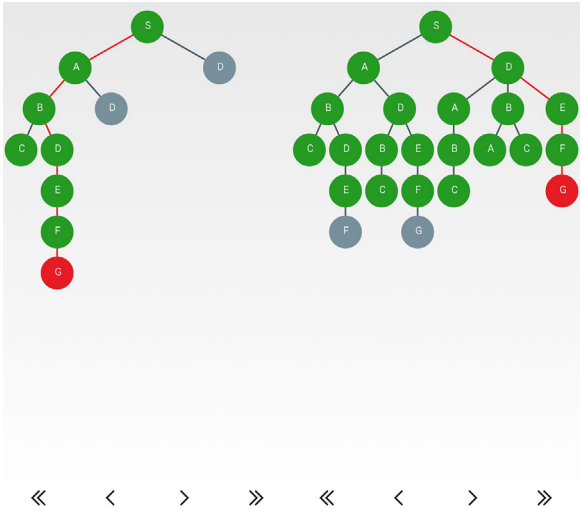


FIGURE 11 Trees for depth-first and breadth-first search simulation

where A, D, and G are the names of the nodes, 10.4, 8.9 and 0 are values of HF, and A-D-5 represent the link with the cost. The link directionality is designated by adding the arrows (<− or −>) instead of the hyphen (−).

The students' aim in this case is to understand the algorithm, that is, the sequence of nodes which are visited in a tree, and the reason why those particular nodes are selected. In contrast to computer simulation version, where comments appear, it is not the case with mobile devices since there is not enough space to display all comments. It is therefore expected that students are well acquainted with theory when doing self-study tasks.

The aim of this particular task is that students learn how to find a path between cities S and G using a particular search algorithm. The road network is given in kilometers (km) and HF is the air distance between a given city and the target city G, as presented in Figure 10.

Depth-first and *breadth-first* methods shown in Figure 11 do not produce the shortest possible distance between cities. If one uses depth-first search, the sequence of nodes is the following: S, A, B, C, D, E, F, G. The located path is 22 km long and connects the following nodes: S-A-B-D-E-F-G. When nodes are visited, there is a lexicographic order. Breadth-first method gives the following path: S-D-E-F-G (19 km).

The *branch and bound* method of tree search is similar to the breadth-first method. This method gives the optimal solution, 17 km long, which goes through the following nodes: S-A-D-E-F-G, as presented in Figure 12.

In case of the *hill-climbing* method, the value of HF is the air distance between cities. In Figure 13 it can be seen that in the first step after the starting node S, node D is selected before node A, the reason being that the HF for node D is 8.9 and for node A 10.4, whereby the lower value of the estimation function is selected. The situation is similar in the following steps of this tree iteration, where for each node the heuristic functions of all subsequent nodes are examined and the one with the lowest value is selected. The finally obtained path of this method is S-D-E-F-G and it is 19 km long.

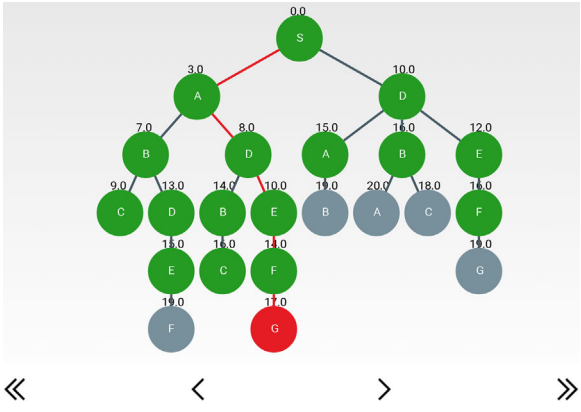


FIGURE 12 Tree for branch and bound search simulation

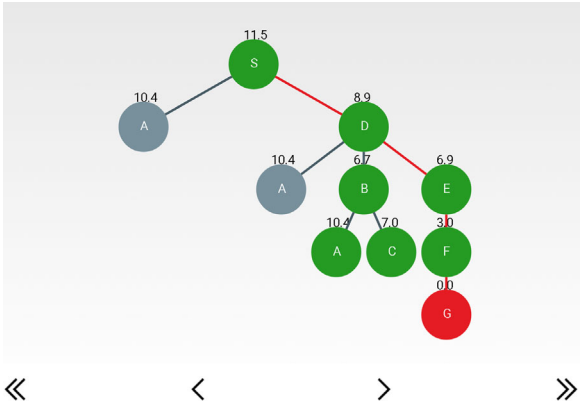


FIGURE 13 Tree for hill-climbing search simulation

Best-first search method is similar to hill-climbing, but this method gives the same solution with far fewer nodes in contrast to the hill-climbing method. The obtained path S-D-E-F-G, 19 km long with only six visited nodes, is shown in Figure 14.

The path obtained using A* method renders the shortest possible path, which is 17 km long (Figure 5). With this group of algorithms the optional use of dynamic programming is also implemented.

4.3 | Gamification—Upgrading SAIL into game

Gamification in education is a process which helps students acquire knowledge through games [15]. Therefore more and more teachers are introducing games into education, using new techniques and tools [6,33]. Contemporary challenges in university education are how to keep students' attention during learning processes [25] and stimulate their interest. What is needed is an user environment which encourages feedback and interaction both between students and teachers, and among students themselves. Researchers also show that proper application of games in computer engineering education would increase the students' knowledge and students' knowledge transferability from academia to the industry [14].

Research has shown that the target population often uses mobile phones to play visual games [29], so one of the upgrades of SAIL is also an animation of algorithms implemented using the *Unity 3D* cross-platform game engine. The implemented game, called "*James Hero*," is based on the already described search algorithms, which can be carried out in parallel so that students can analyze them comparatively. One algorithm is managed by James, and the other one by Shooter. Their goal is to find a way out of a maze and escape from a desert by plane (Figure 15). There are six different maze layouts, i.e. six levels of the game. There are two game modes: an automatic mode of algorithm execution with parallel display, and a gamer mode where the game user can control one of the characters in the game, while the other player aims to prevent the user from escaping the maze and shoots. The first mode is more relevant for students because whenever a character changes position, that change is immediately also shown in the search tree. Figure 16 shows

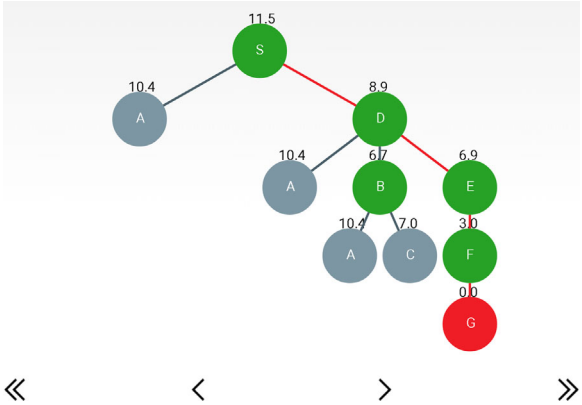


FIGURE 14 Tree for best-first search simulation

FIGURE 15 Upgrading SAIL into game—James Hero game

a game level with a comparative analysis of two search trees: the player called James applies the hill-climbing algorithm, whereas Shooter, the other player who chases him, applies the branch and bound algorithm. It is possible to select a particular kind of search algorithm, adjust the speeds of both players, and give one of the players an advantage of a certain number of steps, which makes the game more adjustable for playing and learning search algorithms.

An advantage of the Unity 3D game engine is that the application can be built for any platform available nowadays, so from the application designed for Android, as an application for Windows and iOS can be easily obtained. Windows and objects are scaled according to the size of the device screen and are displayed correctly. There are more and more games which use this kind of game engine, but they are still not used much in education. There are a few authors in open literature who have implemented one or a couple of algorithms using these development tools, and applied them in education [22,23]. However, the analyzed animations of algorithms are not interactive and designed for learning.

5 | EVALUATION

In 2009/2010 academic year a pilot simulation of search algorithms was developed, with the aim of helping students learn the content better [17]. As a result, the number of points that students acquired when tested in this area was 16.4 out of 20 points, which was a far better score than it had been in the previous three school years, when they scored 8.16–11.81 out of 20 points.

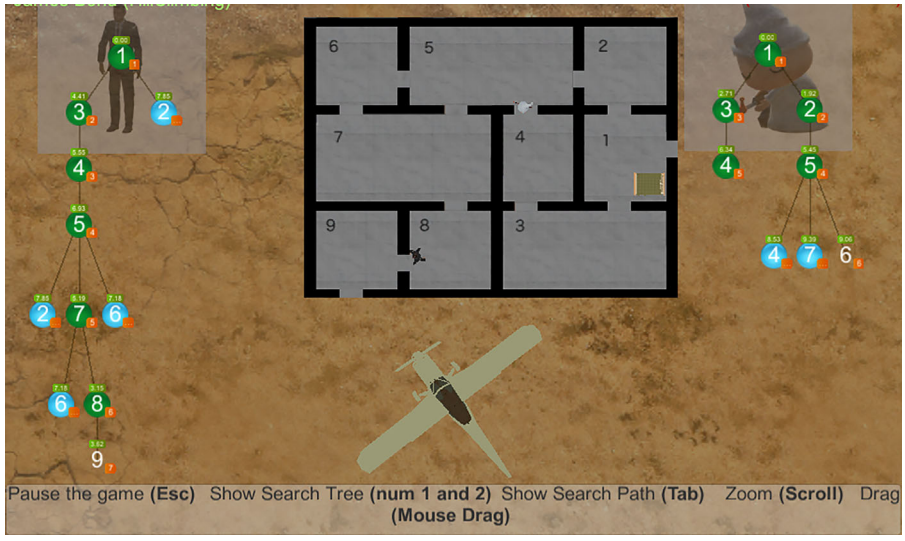


FIGURE 16 Implementation of search algorithms using Unity 3D game engine and a comparative analysis of search algorithms

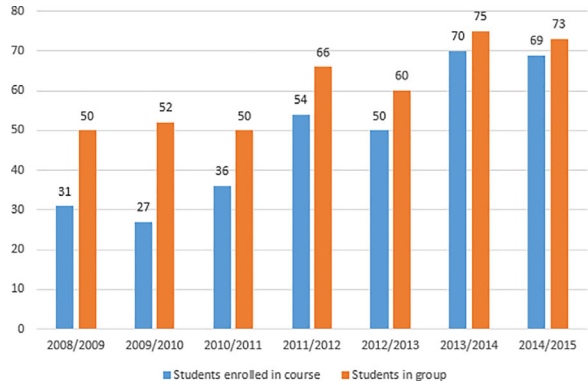


FIGURE 17 The number of students who have chosen elective course Intelligent systems at the Module of SE(in period 2009–2015)

The first version of the software system SAIL, which comprised search algorithms, gaming theory algorithms, formal and fuzzy logic, production systems, GPS and STRIPS algorithms, was used in the summer semester of 2012 [18]. Next year, SAIL was complemented with undetermined environment work algorithms—a system with certainty factors and truth maintenance system, semantic networks and frames, and constraint satisfaction methods. The final version of the system was created in 2013/2014, when induction systems (the ID3 algorithm) and Bayesian networks were added [34]. Students started using the mobile version of SAIL in the summer semester of 2014/2015.

As a result of introducing SAIL into the coursework, it was observed that the number of Software Engineering (SE) students who elected this course gradually increased, as shown in Figure 17. For Computer Engineering (CE) students this course is compulsory, so all students of this module always attended this course.

When it comes to exam performance, it was monitored for students of both study modules (CE and SE) in 2009, the last academic year when no simulation was used, and throughout the 2012–2015 period, when SAIL was upgraded. It should be noted that since 2009 lectures and auditory exercises kept the same format, and what was new were the SAIL lab exercises. Table 5 shows performance in the first exam terms for students of CE, while Table 6 shows performance in the first exam term for students of SE. The first exam term was picked out as the representative one because 60–75% of students take the exam in this exam term.

In addition, we analyzed the number of students who passed the Intelligent systems course, across all six exam terms: June, July, September, October, January, and February. The number of CE students who passed successfully before SAIL was introduced was under 80%, whereas after SAIL was introduced, this number was 83–92%. Before SAIL, exam success of SE students was maximum 80% in all exam terms, whereas after sail was introduced, the number was 88–90%, as shown in Table 7. Similarly, the average number of points scored in the final exam shows that exam success improved owing to visual simulations in class and self-study at home using SAIL.

In 2009/2010, the homework which carried 20% of the final number of course points was introduced into the course. Before it was introduced, students had to make these 20% by answering theoretical questions, and since 2009/2010 students could choose whether they answer a theoretical question in the final exam or do homework instead.

TABLE 5 The course scores in June 2009 and in period 2012–2015 (Computer Engineering Module)

	Exam period				
Percentage range of scored points	**June 2009**	**June 2012**	**June 2013**	**June 2014**	**June 2015**
90–100%	5.31%	7.27%	14.81%	8.47%	12.3%
80–89.99%	8.85%	10%	8.33%	14.41%	11.47%
70–79.99%	16.81%	10%	11.11%	9.32%	7.38%
60–69.99%	20.35%	11.82%	4.63%	14.41%	17.21%
50–59.99%	9.74%	10%	14.81%	15.25%	9.84%
Below 50%	9.74%	12.73%	7.41%	9.32%	7.38%
Average number of points	63.5	68.8	76.3	72.5	75.25
% of students who missed the exam	29.20%	38.18%	38.9%	28.82%	34.42%

TABLE 6 The course scores in June 2009 and in period 2012–2015 (Software Engineering Module)

	Exam period				
Percentage range of scored points	**June 2009**	**June 2012**	**June 2013**	**June 2014**	**June 2015**
90–100%	16.13%	22.22%	20%	12.86%	15.94%
80–89.99%	3.22%	7.41%	12%	11.43%	11.59%
70–79.99%	9.68%	5.56%	10%	12.86%	11.59%
60–69.99%	16.13%	3.70%	10%	11.43%	10.14%
50–59.99%	9.68%	1.85%	6%	10%	8.7%
Below 50%	12.90%	3.70%	4%	12.86%	8.7%
Average number of points	68.2	81.5	79.25	80.33	81.86
% of students who missed the exam	32.26%	55.56%	38%	28.57%	33.33%

Homework is based on the implementation of an algorithm with the existing frames of graphic user interface, realized in the programming language Java. Some of the problems which students solved for homework were: Block World, Nine Men's Morris, Freedom, Reversi (Othello), Wumpus world, Backgammon. Their goal was to write an algorithm which would solve the problem with the best possible execution performance of the given algorithm. At the end of homework presentation, there is a tournament which means that all students compete with their solutions, and three best students get bonus points. Homework is very significant for students, because they learn and implement algorithms, acknowledge their advantages and disadvantages, connect their program code with classes available in task formulation, and implement a complete game based on algorithms. In this way gamification is applied through course homework.

Table 8 shows that the number of students who do homework increased since 2011/2012, which coincides with the introduction of first simulations in SAIL.

The analysis also included a student poll for students of both study modules (CE and SE) who attended the course in 2013/2014 and 2014/2015. About 80% of students who took the course and used SAIL took part in the anonymous poll, which consisted of six questions about SAIL:

1. (Q1) Is the user interface of SAIL intuitive and user-friendly?
2. (Q2) Rate the level of interaction of the user and the system, the possibility to make your own examples, and load the existing ones from the task collection.
3. (Q3) Mark the possibility to start the step-by-step simulation for each algorithm and observe the simulation in SAIL.
4. (Q4) To what extent did you find the lab exercises helpful in exam/mid-term exam preparation?
5. (Q5) Was SAIL helpful for self-study and mid-term exam/exam preparation?
6. (Q6) Your general impression of SAIL.

Students answered these questions on the 1 to 5 scale: 5, very satisfied; 4, satisfied; 3, average; 2, unsatisfied; 1, very unsatisfied. The average rating per question was presented in Table 9. The rating is very high for question 4, where they evaluated the impact of lab exercises, and for question 6, where they marked the general impression of SAIL Students also expressed satisfaction when answering questions related to user interface and user-friendliness, level of interaction with the system, and observing the simulation step-by-step. The lowest rating was for question 5, which relates to self-study and exam preparation using SAIL, because the system could not completely replace the traditional course material, textbooks and slides. However, it can be noted that the rating was somewhat higher in 2014 and 2015, after the mobile version of SAIL was introduced. The survey showed that about 55% of students use the mobile application to learn.

The analysis also considered the time that students needed to prepare for the exam before and after SAIL was introduced. The questions were:

TABLE 7 The course passing rate for the whole school year before and after the introduction of a software system SAIL

	School year				
Study module	**2008/09**	**2011/12**	**2012/13**	**2013/14**	**2014/15**
Computer engineering	86/113 76.22%	101/110 91.82%	90/108 83.33%	99/118 83.89%	109/122 89.34%
Software engineering	23/31 74.19%	51/54 94.44%	48/50 96.0%	62/70 88.57%	65/69 94.2%

The m/n ratio indicates the number of students who passed the exam and the number of students who were eligible to take the exam.

TABLE 8 The number of students who has finished homework compare to the total number of students who enrolled course

	Exam period					
Study module	**June 2010**	**June 2011**	**June 2012**	**June 2013**	**June 2014**	**June 2015**
Computer engineering	21/113	29/107	36/110	40/108	45/118	57/122
Software engineering	8/27	11/36	30/54	32/50	36/70	37/69

TABLE 9 The results from the student survey in school year 2013/2014 and 2014/2015

	Survey period	
Question	**June 2014**	**June 2015**
Question #1	4.26	4.18
Question #2	4.12	4.14
Question #3	3.98	4.08
Question #4	4.46	4.44
Question #5	3.33	3.99
Question #6	4.42	4.40

1. (Q7) How much time did you invest into this course this year (lessons, lab exercises, homework)?
2. (Q8) How much time did you invest into self-study to prepare for the exam?

Results are presented in Table 10. Notably students using SAIL needed less time to prepare for the exam and they invested less time in self-study for the exam. The school accreditation process also involved student polls, so it was an opportunity to analyze the course load expressed through ECTS (*European Credit Transfer and Accumulation System*). Before SAIL and the homework were introduced, the estimated course load for Intelligent systems was 6.55 ECTS for CE students, and 7.02 ECTS for SE students in 2008. After SAIL was introduced, the last poll in June 2015 showed the estimated number of ECTS for this subject was 5.45 for CE students, and 5.52 for SE students. This analysis shows that the number of ECTS points is appropriate for the workload of the Intelligent systems course.

Table 11 also shows student evaluation of professor's and teaching assistant's work. All students participated in all surveys, because a student survey was mandatory, at the School of EE at the University of Belgrade, after the course was over. It is interesting to highlight a couple of additional comments of students who positively evaluated SAIL within the student poll:

1. Subject content is very interesting but it is much clearer thanks to SAIL.
2. The implemented system is truly great because it gives detailed explanations and can help students in case they miss a lesson.
3. The way computers are used for algorithm simulation in this course is excellent and should be introduced into other courses as well.
4. The software tool SAIL is very useful because as students we can study not only from slides, but interactively using a computer, and check whether our solution is correct.

TABLE 10 Student learning time

	Question #7			
Study module (year of the survey)	**Less than 40 hr (%)**	**40–89 hr (%)**	**90–180 hr (%)**	**More than 180 hr (%)**
CE (2009)	43.40	45.28	9.43	1.89
CE (2015)	54.54	37.88	7.58	0
SE (2009)	36.59	41.46	14.63	7.32
SE (2015)	40.38	48.08	9.62	1.92
	Question #8			
Study module (year of the survey)	**Less than 7 days (%)**	**8–15 days (%)**	**16–30 days (%)**	**More than 30 days (%)**
CE (2009)	49.06	45.28	1.89	3.77
CE (2015)	60.61	34.85	4.54	0
SE (2009)	39.02	39.02	14.64	7.32
SE (2015)	55.77	36.54	7.69	0

TABLE 11 Assessments of teachers on student survey

	Average rating Rate on a scale from 1 to 5					
Study module	**June 2013 [P]**	**June 2013 [TA]**	**June 2014 [P]**	**June 2014 [TA]**	**June 2015 [P]**	**June 2015 [TA]**
Computer engineering	4.59	4.36	4.4	4.49	4.61	4.78
Software engineering	4.16	3.92	4.41	4.16	4.7	4.55

P, professor; TA, teaching assistant.

6 | CONCLUSION

Artificial intelligence as a field comprises a large number of very complex algorithms used for solving various problems. Since those systems are more and more often used in education and for everyday problem solving, there is a need for a system which would enable learning and simulation of such algorithms. This paper presented a software system called SAIL, which was created at the Department of Computer Engineering and Information Theory of the School of Electrical Engineering University of Belgrade. The aim of the system is to present algorithm simulation and strategies from the artificial intelligence field: search algorithms and gaming theory algorithms, formal and fuzzy logic, production systems, GPS and STRIPS algorithms, semantic networks and frames, algorithms for undetermined environment operating systems, induction algorithms, and Bayesian networks. There are independent simulations and educational tools implemented at other universities around the world. None of them, however, comprises all these areas and none of them shows a solution that is detailed enough. The missing part is the step-by-step execution of algorithms, whereby each step is analyzed, which is necessary so that students would grasp course content better. In addition, none of the analyzed software systems for learning and visualization of AI algorithms had a mobile application convenient for self-study.

The use of SAIL in education helps students understand the content and keep up with tasks and defined problems as efficiently as possible, while at the same time teachers can rely on visual representation and interactive work with students when teaching these concepts, which are abstract, complex, and difficult to explain and understand. Students can carry out algorithm simulations with existing examples presented in lessons and lab exercises, but they can also do simulations for their own problems using various conditions and input parameters.

In this way the analysis of algorithms from the field of artificial intelligence becomes a lot more systematic and rational, so students can quickly and easily learn the steps of implemented algorithms and derive conclusions. The advantages for students were presented through various analyses and numerical indicators obtained in the period 2009–2015, when the research was carried out. Some of the parameters we used to verify this research were: the increased number of software engineering students who choose to take the course, the increased number of students who opt-in to do homework rather than theoretical questions in the final exam, better exam success in the first exam term and generally in all exam terms during school year, higher exam score, less time invested into studying and exam preparation but still in accordance with the number of ECTS credits allotted to this course, higher evaluation of teachers in student polls after simulations and lab exercises were introduced.

This research also shows that it is possible to upgrade the implemented system into a game using a game engine. In this way, using better interface graphics and complying with the principles of gamification in education, students could learn the content and simulate the work of algorithms through a real computer game. If the modular system SAIL were upgraded, the system for learning and application of artificial intelligence algorithms could very easily find application in other fields, as an intelligent system used for solving various problems.

ORCID

Drazen Draskovic http://orcid.org/0000-0003-2564-4526

REFERENCES

1. AI Exploratorium software system [Online]. Available: http://webdocs.cs.ualberta.ca/~aixplore/, accessed on May, 5th, 2013.
2. AI search simulation [Online]. Available: http://www.cs.rmit.edu.au/AI-Search/, accessed on March 1st, 2016.
3. AIMA3e software system [Online]. Available: http://aima-java.googlecode.com/svn/trunk/aima-all/release/aima3ejavademos.html, accessed on March, 28th, 2015.
4. AIspace software system [Online]. Available: http://www.aispace.org/, accessed on March 15th, 2016.
5. S. Amershi et al., *Designing CIspace: Pedagogy and usability in a learning environment for AI*, in Proc. 10th Annu. SIGCSE conf. on Innovation and technology in Comp. Sci. Educ., vol. 37, issue 3, Sept. 2005, pp. 178–182.
6. S. Borges et al., *A systematic mapping on gamification applied to education*, in Proc. 29th Annu. ACM Symp. on Applied Computing, 2014, pp. 216–222.

7. I. Borm and L. Rothkrantz, *Teaching artificial intelligence techniques using multi-agent soccer*, in Proc. 3rd E-learning conf., Coimbra, Portugal, Sept. 2006 [Online]. Available: http://elconf06.dei.uc.pt/pdfs/paper34.pdf
8. V. Ciesielski and P. McDonald, *Using animation of state space algorithms to overcome student learning difficulties*, in Proc. 6th Annu. Conf. on Innovation and Technology in Computer Science Education, Canterbury, UK, June 2001, pp. 97–100.
9. Computer Science Curricula 2013—Curriculum Guidelines for Undergraduate Degree Programs in Computer Science [Online]. Available: https://www.acm.org/education/CS2013-final-report.pdf, accessed on July 11th, 2015.
10. Computing Curricula 2005—The Overview Report covering undergradute degree programs in CE, CS, IS, IT and SE [Online]. Available: http://www.acm.org/education/education/curric_vols/CC2005-March06Final.pdf, accessed on September 18th, 2015.
11. Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering [Online]. Available: http://www.acm.org/education/curric_vols/CE-Final-Report.pdf, accessed on September 16th, 2015.
12. M. Cvetanovic et al., *ADVICE—Educational System for Teaching Database Courses*, IEEE Trans. Educ. **54** (2011), 398–409.
13. W. Da Silva Lourenco, S. De Araujo Lima, and S. Alves De Araujo, *TASNOP: A tool for teaching algorithms to solve network optimization problems*, Comput. Appl. Eng. Educ. [online], (2017), 1–10. https://doi.org/10.1002/cae.21864
14. A. Deshpande and S. Huang, *Simulation games in engineering education: A state-of-the-art review*, Comput. Appl. Eng. Educ. **19** (2009), 399–410.
15. S. Deterding et al., *From game design elements to gamefulness: defining 'gamification'*, in Proc. 15th ACM Int. Academic Mind-Trek Conf., Tampere, Finland, 2011, pp. 9–15.
16. J. Djordjevic et al., *CAL: Computer aided learning in computer architecture laboratory*, Comput. Appl. Eng. Educ. **16** (2008), 172–188.
17. D. Draskovic, V. Batanovic, and B. Nikolic, *Software system for expert systems learning*, (in Serbian), in Proc. 18th Telecommunications Forum TELFOR 2010, Belgrade, Serbia, pp. 1129–1132.
18. D. Draskovic and B. Nikolic, *Software system for expert systems learning*, in Proc. IEEE Int. Conf. AFRICON 2013, Pointe-Aux-Piments Mauritius, 2013, pp. 1–6.
19. T. Ellis, *Animating to build higher cognitive understanding: A model for studying multimedia effectiveness in education*, J. Eng. Educ. **93** (2004), 59–64.
20. Fuzzy Logic Library [Online]. Available: http://ffll.sourceforge.net/fcl.htm, accessed on September 16st, 2011.
21. M. Hall et al., *The WEKA data mining software: An update*, ACM SIGKDD Explorations Newslett. **11** (2009), 10–18.
22. N. Harshfield, D. Chang, and R. Ragade, *A unity 3D framework for algorithm animation*, in Proc. 20th Int. Conf. on Computer Games, Louisville, KY, USA, July 2015, pp. 50–56.
23. Z. He, M. Shi, and C. Li, *Research and application of path-finding algorithm based on unity 3D*, in Proc. IEEE 15th Int. Conf. Computer and Information Science, Okayama, Japan, June 2016.
24. R. Hill and A. van den Hengel, *Experiences with simulated robot soccer as a teaching tool*, in Proc. 3rd Int. Conf. on Information Technol. and Applicat., vol. 1, 2005, pp. 387–390.
25. M. Ibanez, A. Di-Serio, and C. Delgado-Kloos, *Gamification for engaging computer science students in learning activities: A case study*, IEEE Trans. Learning Technologies **7** (2014), 291–301.
26. H. Jabbar and R. Khan, *Survey on development of expert system from 2010 to 2015*, in: Proc. 2nd Int. Conf. on Information and Communication Technology for Competitive Strategies, article no. 130, Udaipur, India, Mar. 2016.
27. JUNG API, Javadoc [Online]. Available: http://jung.sourceforge.net/, accessed on July 22nd, 2010.
28. U. Kose and D. Koc, *Artificial intelligence applications in distance education*, 1st ed., IGI Global, US, 2014.
29. Q. Liu, L. Diao, and G. Tu, *The application of artificial intelligence in mobile learning*, in Proc. Int. Conf. on System Science, Engineering Design and Manufacturing Informatization, Yichang, China, 2010, pp. 80–83.
30. Z. Markov et al., *Enhancing undergraduate AI courses through machine learning projects*, in Proc. 35th Annu. Conf. Frontiers in Educ., 2005, pp. T3E-21.
31. R. E. Mayer and R. Moreno, *A cognitive theory of multimedia learning: implications for design principles*, in Proc. ACM SIGCHI Conf. on Human Factors in Computing Systems, 1998.
32. P. McDonald and V. Ciesielski, *Design and evaluation of an algorithm animation of state space search methods*, J. Comp. Sci. Educ. **12** (2010), 301–324.
33. A. McGovern, Z. Tidwell, and D. Rushing, *Teaching introductory artificial intelligence through Java-Based Games*, in Proc. 2nd Symp. on Edu. Advances in Artificial Intelligence, 2011, pp. 1729–1736.
34. K. Milenkovic, D. Draskovic, and B. Nikolic, *Educational software system for reasoning and decision making using Bayesian networks*, in Proc. IEEE Global Eng. Educ. Conf. EDUCON 2014, Istanbul, Turkey, 2014, pp. 1–6.
35. MIT Open Course Ware software system (course: Artificial Intelligence, Massachusetts Intitute of Technology, US, Prof. P.H. Winston) [Online]. Available: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010/demonstrations/, accessed on March 1st, 2016.
36. M. Pantic, R. Zwitserloot, and R. Grootjans, *Teaching introductory artificial intelligence using a simple agent framework*, IEEE Trans. Educ. **48** (2005), 382–390.
37. I. Russell et al., *MLeXAI: A project-based application-Oriented model*, J. ACM Trans. Comput. Educ. **10** (2010), 1–36.
38. S. Russell and P. Norvig, *Artificial intelligence: A modern approach*, 3rd ed., Prentice Hall, US, 2009.
39. Searcher software system [Online]. Available: http://www.csc.liv.ac.uk/~cs8js/code/coffeescript/project-testing/, accessed on May, 5th, 2013.
40. Simple expert system simulation [Online]. Available: http://lucy.ukc.ac.uk/ExpertSys/ExpertSys.html, accessed on May, 5th, 2013.
41. Simulator Sudoku Solver [Online]. Available: http://sudoku.unl.edu/SudokuSet/SudokuSetV2/index2.html, accessed on May, 5th, 2013.
42. Z. Stanisavljevic et al., *COALA—System for visual representation of cryptography algorithms*, IEEE Trans. Learn. Technol. **7** (2014), 178–190.
43. The Chinese Room simulation, The Mind Project, Illinois State University, US [Online]. Available: http://www.mind.ilstu.edu/curriculum/modOverview.php?modGUI=203, accessed on March 1st, 2016.
44. I. H. Witten et al., Weka: Practical machine learning tools and techniques with Java implementations, University of Waikato, Hamilton, New Zealand, Aug. 1999 [Online]. Available: http://researchcommons.waikato.ac.nz/bitstream/handle/10289/1040/uow-cs-wp-1999-11.pdf?sequence=1&isAllowed=y, accessed on August 22nd, 2016.

D. DRASKOVIC received his BSc (2009) and MSc (2011) degrees in Software Engineering from the University of Belgrade, School of Electrical Engineering, Serbia, where he is currently working toward his PhD in Software Engineering. Since 2010, he has been a Teaching and Research Assistant at the University of Belgrade. He teaches several courses on software testing, software engineering, software project management, internet programming, intelligent systems and data mining. His research interests are within the area of artificial intelligence and include searching algorithms, advanced reasoning techniques, machine learning and data analysis. He is a member of IEEE from 2013.

M. CVETANOVIC received his BSc (2003), MSc (2006) and PhD (2012) degrees in Electrical and Computer Engineering from the University of Belgrade, School of Electrical Engineering, Serbia. He is currently an Associate Professor at the University of Belgrade. He teaches several courses on databases and database software tools, information systems, software engineering of large databases, e-business infrastructure and programming mobile applications. His research interests include the area of database and information systems, artificial intelligence, big data and reverse engineering.

B. NIKOLIC received his BSc (1996), MSc (2001) and PhD (2005) degrees in Electrical and Computer Engineering from the University of Belgrade, School of Electrical Engineering, Serbia. He is currently a Full Professor at the University of Belgrade. He teaches several courses on intelligent systems, internet programming, data mining, and natural language processing. His research interests include the area of algorithms and data structures, artificial intelligence, web programming, software design and engineering education.

How to cite this article: Draskovic D, Cvetanovic M, Nikolic B. SAIL—Software system for learning AI algorithms. *Comput Appl Eng Educ*. 2018; 26:1195–1216. https://doi.org/10.1002/cae.21988