MODELLING OF INTELLIGENT AGENTS USING A–PROLOG

Fernando Zacarias[1], Rosalba Cuapa[2], Luna Jimenez[1] and Noemi Vazquez[2]

[1]Computer Science, Benemérita Unversidad Autónoma de Puebla, Puebla, México
[2]Faculty of Architecture, BUAP, Puebla, México

ABSTRACT

Nowadays, research in artificial intelligence has widely grown in areas such as knowledge representation, goal-directed behaviour and knowledge reusability, all of them directly relevant to improving intelligent agents in computer games. In a particular way, we focus on the development of a novel algorithm that allow an agent to combine fundamental theories such as reasoning, learning and simulation. This algorithm combines a system of logical rules and a simulation mechanism based on learning that makes our agent has an infallible mechanism for decision-making into the game "connect four". The logic system is developed in a modelling language known as Answer Set Programming or A–Prolog. This paradigm is the integration of two well-known languages, namely Prolog and ASP.

KEYWORDS

Agents, Games, A–Prolog, Answer Set Programming & Logic Programming

1. INTRODUCTION

When we want to design an intelligent agent capable of behaving intelligently in some environment, then we need to supply this agent with sufficient knowledge about this environment. Thus, we need a modelling language that provides a well defined, general, and rigorous framework for expressing this knowledge, together with some precise and well understood way of manipulating sets of sentences of the language which will allow us to draw inferences, answer queries, and update both the knowledge base and the desired program behaviour. There are several proposals that cover some of these characteristics [12], [13], [14], all of them based on Answer set programming (ASP). Answer Set Programming (ASP, Stable Logic Programming or A–Prolog [1]), is the realization of much theoretical work on Non-monotonic Reasoning and Artificial Intelligence applications of Logic Programming. ASP is a novel paradigm of logic programming that has great acceptance in the artificial intelligence community. One of the most important reasons for its acceptance is the existence of efficient software to compute answer sets [2].

The computational logic provides efficient solutions, at a sufficient level of abstraction afforded by the nature of its very foundation in logic. Particularly, we can to say that the computational logic has its major asset in substance, method and theories, as mentioned in [2]. Next, we consider the next interpretation under Answer Set Programming [1], [2]. Let T be a given theory, its knowledge is understood as a set of formulas F such that these are derived in T using intuitionistic logic. This makes sense, since in intuitionistic logic according to Brouwer [3], F is identified as: "I know F" (some reader would prefer to understand the notion of "knowledge" as "justified belief", in the context of models). An agent whose knowledge base is the theory T, believes F if and only if F belongs to every intuitionistically complete and consistent extension of

T by adding only negated literals (here, "belief" could be better interpreted as "coherent" belief). In a way we agree on what Kowalski affirms, "Logic and Logic Programming need to be put into place: Logic within the thinking component of the observation–thought–action cycle of a single agent, and logic programming within the belief component of thought".

2. A–PROLOG AS MODELLING LANGUAGE

A–prolog provides a simple, expressive and efficient language that can be well suited for modelling the agent rational component into computer science and artificial intelligence. The answer sets for a logic program can be described as the satisfying interpretations for a set of propositional formulae.

2.1. Propositional Logic

We use the language of propositional logic in order to describe rules within logic programs. Formally we consider a language built from an alphabet consisting of atoms: p0, p1, ...; connectives: $\wedge$, $\vee$, $\leftarrow$, $\bot$; and auxiliary symbols: '(', ')', '.', where $\wedge$, $\vee$, $\leftarrow$ are 2-place connectives and $\bot$ is a 0-place connective. Formulas are defined as usual. The formula $\top$ is introduced as an abbreviation of $\bot\leftarrow\bot$, not F as an abbreviation of $\bot \leftarrow$ F, and F $\leftrightarrow$ G as an abbreviation of (G $\leftarrow$ F) $\wedge$ (F $\leftarrow$ G). The formula F $\rightarrow$ G is another way of writing the formula G $\leftarrow$ F, we use the second form because of tradition in the context of logic programming. We will represent the default negation with $\neg$. A signature L is a finite set of atoms. If F is a formula then the signature of F, denoted as LF, is the set of atoms that occur in F. A literal is either an atom a (a positive literal) or a negated atom $\neg$a (a negative literal). A logic program is a finite set of formulas. The syntax of formulas within logic programs has been usually restricted to clauses with a very simple structure. A clause is, in general, a formula of the form H $\leftarrow$ B where H and B are known as the head and body of the clause respectively. Two particular cases of clauses are facts, of the form H $\leftarrow \top$, and constraints, $\bot \leftarrow$ B. Facts and constraints are sometimes written as H and $\leftarrow$ B respectively.

3. INTELLIGENT AGENTS

The agent technology is a technique widely used in artificial intelligence in the last twenty years. Intelligent agents comprise individual abstract entities that can acts on their own and can to decisions making based on its experiences [9], [15], [16], [18], [19]. Integrating these autonomous intelligent agents into software development substantial improvement problem solving. Furthermore, our intelligent agents possess learning, reasoning and self-organized capability (see figure 1). We have incorporated an intelligent agent in our study due to their highly reliability and competence in solving problems. This intelligent agent provide a medium of deploying an expert in our particular application.

In the figure 2, we show the agent reaction based on its perception and past games (saved in files) programed by the developer. Intelligent agent perceives and acts on the environment deployed.

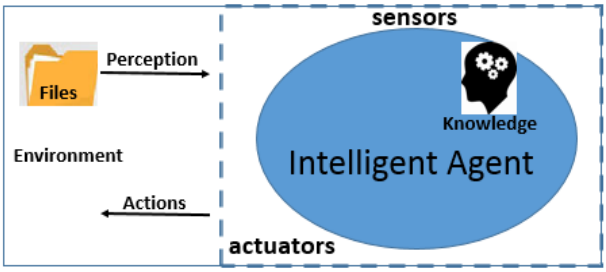

Figure 1. Deatiled structure of an intelligent agent

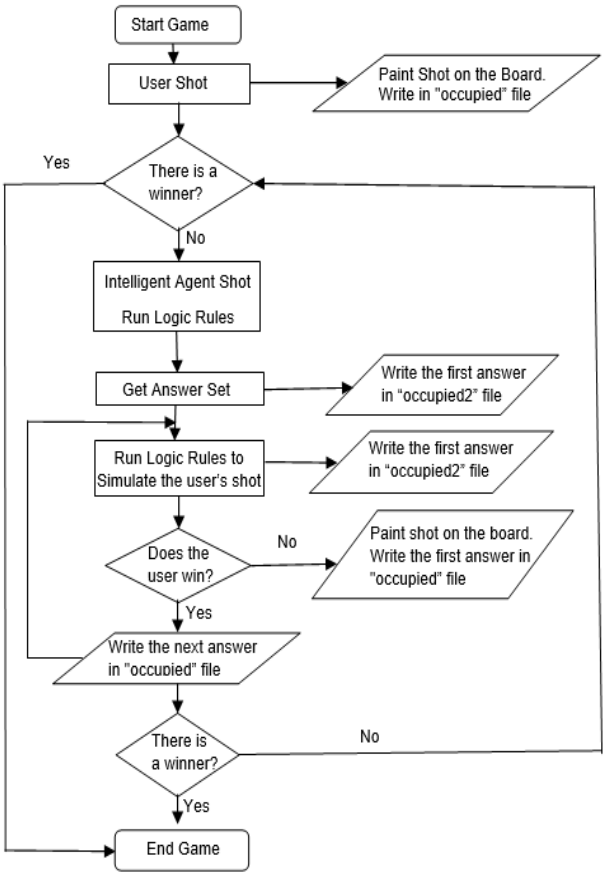

Figure 2. The Agent Simulation Process

In the figure 2, we show the agent reaction based on its perception and past games (saved in files) programed by the developer. Intelligent agent perceives and acts on the environment deployed.

4. CONNECT FOUR

Connect four is a two-player connection game in which the players first choose a color and then take turns dropping colored discs from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the next available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs [4]. For classic Connect Four played on 6 high, 7 wide grid, there are

4,531,985,219,092 positions [5] for all game boards populated with 0 to 42 pieces. There are several proposals on how to address this problem however, many of these do not guarantee winning. Many people know the rules of the game, but do not know what strategy to play. Thus, in this paper we fill this gap through an agent that covers it through a system of rules combined with a simulation algorithm.

There are some strategies [6], [17] that players can execute to win a game, although they do not ensure that this is the case, for example:

- It is suggested that the game begin in the middle row. This will give your opponent less chance to make a line.
- Plan your movements thinking about what your opponent will do.
- Organize your moves in a way that forces your opponent to throw in a certain place. This can be used for own benefit because you can use these moves as a ladder to get to where your pieces are and make a line.
- In the same way, try to avoid that your pieces serve as a ladder for your opponent.
- Work several lines at the same time, this will give you more options so it will increase your chances of winning.

5. LEARNING

Case based reasoning is an artificial intelligence technique used by many researchers since the eighties. Case based reasoning utilizes a set of cases stored in files, which is the main source of knowledge. This method is based on the idea of making use of previous experience and selecting the most similar previous case to resolve present problem this is because similar problems would definitely have similar answers. Although in our particular case, finding the most similar case to the present problem is not to repeat the same error in the decision making process.

In this study, we use case based reasoning as a problem solving technique that deploys reasoning rules which utilizes similarity search in the file bank whose contents are examples of games previously played in which the player lost that game. These examples are experiences and previously successful cases for addressing present problems. Where all cases are indexed with real data using a database structure based on files for quick and accurate information. Thus, case based reasoning makes reasoning through knowledge and experience, which are utilized as solutions to similar case problems.

In figure 3, we show the general structure of case based reasoning technique. This technique is used to retrieve previous similar case. With this case, the agent generates a similar rule considering not provoking the same error previously produced; guaranteeing that the new decision (the new move) is different to those previously made.

It is important to note that in our development the application of the case based reasoning technique is simple and clear. The agent generates a new rule by changing only the position of the square chosen in the previous similar game found in the base of cases corresponding to the last movement in which he lost the game. To increase the robustness of the intelligent agent based on the case-based reasoning technique, we must continue developing each of the parts that make up the general structure of this technique [10].

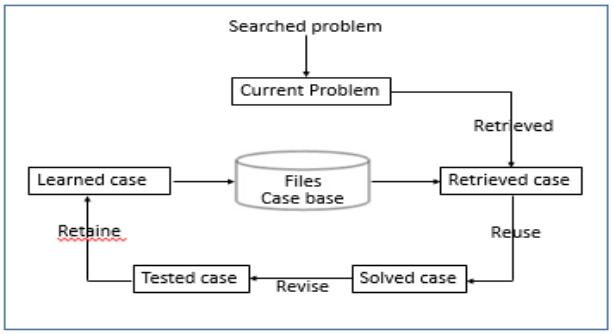

Figure 3. Case Based Reasoning Cycle

6. INFALLIBLE AGENT VIA LEARNING

For the modelling of the intelligent agent, three fundamental actions are defined that the agent must execute, as well as a simulation process that will lead the agent to make the best decision (the simulation is to anticipate a bad decision, see figure 3).

1. The first of these actions is observation, as we have already mentioned, we need to supply this agent with sufficient knowledge about his environment. This agent is able to observe when he knows the current configuration of the game board, the occupied and unoccupied squares.

2. The second action of this agent is to update, once it has observed how his environment is then update your information to move to the next action.

3. The last action that the agent executes is the reasoning, with the information obtained from the observation and after updating what he knew then he makes a decision that will allow him to execute a response to his environment.

The simulation process has been modelled with the purpose of the agent learning to recognize the patterns in which the game may be lost (the simulation is to anticipate a bad decision and that the agent can choose another movement). In the simulation process, the agent analyses two more levels of depth to determine what is the error committed and in this way to correct the action. Then, the agent performs a backtracking process to change the action chosen by the following answer set generated by A-Prolog. The actions that the intelligent agent executes depend of the logic rules system, this system is developed in a modelling language known as answer set programming (dlv system or A–Prolog) [7]. This system has proven to offer efficiency in the modelling and execution of complex problems.

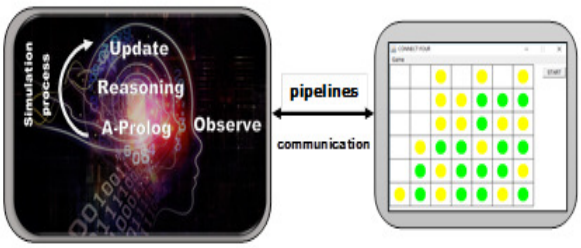

Figure 4. Smart agent based on A–prolog embedded in our game

7. LOGIC RULES SYSTEM

A logic rule system is a finite set of formulae, for the modelling of this system, the board of the game connect four can be seen as a set of squares, each of these is assigned a coordinate formed by a value on the X axis and another on the Y axis. The values that variable X can take varies from zero to six and the variable Y is from zero to five. Another variable used for modeling the system is the color of the discs assigned to each player. We use two colors: yellow and green; for the definition of formulas only the first letter of each color is used, in case of the square is empty, we use the letter 'e'.

In the language of propositional logic, the description of the state of a square is considered a fact, here is an example, which denotes that the square whose coordinates are X = 3 and Y = 0 is occupied by a green disc:

square (3,0, g).

To describe the configuration of the board, the intelligent agent reads from a text file the description of each of the occupied and unoccupied squares (knowledge as facts). The agent based on the update of the information that has been observed, the logical rules system is executed in order to choose its best movement. This logical rules system is divided into three subsets: win, block and build.

To define the rules of the system, clauses are defined with a simple structure of the form H ← B, for our modeling, H represents the formula that the agent uses to generate its set of solutions and B are the facts that are obtained from the observation-update, serve for the construction of the clause.

```
w(X, Y, y) :- square(X, Y, e),
              square(Z, A, y), X=Z+1, Y=A+1,
              square(W, B, y), W=X+1, B=Y+1,
              square(R, C, y), R=X+2. C=Y+2.
```

This code segment is an example of a clause of the winning subset since the head of the clause starts with the letter "w" (winner). For our agent, that clause means that it can execute a winning action in the X, Y coordinates with the disc of yellow color, this if the facts of the body of the clause are satisfy. This means that the square to occupy is empty and the others occupied squares form a line of four discs of the same color. Each set of formulae verifies the squares that are occupied and those that are empty. The clauses to obtain the winning actions must satisfy that three squares are in horizontal, vertical or diagonal line and that the fourth disc can be placed in that same line, i.e., the clause is satisfied when it is possible to place four discs in a line.

The set of rules to execute the action of blocked is modeled in the same way as the rules to win. In this case, the clause is satisfied when the facts of the clause body are satisfied. This is the case where the opponent has placed three disks in line, then, the agent blocks that play by placing a disc in the proper coordinate.

When our agent must build his game or develop strategies that lead him to win, he must validate that he cannot win or block. Therefore, it is said that it can occupy a square to start the construction of a game, in other words, when the agent does not recognize any configuration to block or win then builds a game to achieve his goal. The clauses for the modeling of this action have the same structure as the previous two. As part of this observation, the agent is able to recognize if there is a winning player. To execute that action is also model a set of clauses, with

the same structure as the actions to win, block and build, the body of these clauses is satisfied when four squares have been found with disks of the same color in a horizontal, diagonal or vertical line.

The communication between the agent and the user (user-interface or environment) is done through pipelines (in a similar way to what unix does). The information of occupied and unoccupied squares is obtained when the agent needs to observe, that information is what fills the formulas modeled in the body of the clauses of the logical rules system. Then, the agent updates what he knows about his surroundings based on new information obtained from observation. In this way, the rule system will be executed with updated knowledge. The logical rules system modeled in A-Prolog provides our agent with a set of answers sets, which indicate the action to be executed (win, block, build), then we say that the agent has reasoned, thus fulfilling its basic cycle.

To make our agent capable of learning, it is necessary to introduce a new action, namely, the simulation process. This simulation process will allow our agent to anticipate possible erroneous movements that lead him to lose the game. However, if even with this process the agent loses, it is at this point that the agent will save the entire game in order to learn from it through the technique known as case-based reasoning.

8. SIMULATION PROCESS

One of the strategies we use when we play connect four is to plan our movements thinking about what your opponent will do. This strategy is the basis for the modeling of the simulation process, i.e., the agent is able to simulate the next action of his opponent and observe if the configuration generated causes that our agent to lose. If that happens then he must decide to place his disc in other coordinates different from those he had used before simulating the action of his opposing player. For this, our agent will choose the next answer set generated by A–Prolog. Furthermore, if there are no more answer sets, the agent applies case based reasoning (described in section V. Learning), that is, applies the rule generated from the game that is most similar to the current one. In this way, the agent guarantees that the configuration generated with his new choice will not lead him to a failure already foreseen. The algorithm designed to implement the simulation process is as follows:

```
Start <Simulation Process>
if (turn_agent)
execute A–Prolog<logic_rules_system.txt>
            <observation_file.txt><update_file.txt>
   read<answer_set_file.txt>
   get <X,Y>
         update <square_occupied.txt>
execute A–Prolog<logic_rules_system_user.txt>
            get <X,Y> for simulated action user
                  update <square_occupied.txt>
                  execute A–Prolog<win.txt>
                  if(user==win)
                  {
                    X←X1
                    Y←Y1
                    square(X1,Y1,y)
                  }
                  else
```

```
                {
                  X←X
                  Y←X
                  square(X1,Y1,y)
                }
execute A–Prolog<win.txt>
if (agent==win)
end Simulation process
else
   (turn_user)
End Simulation Process
```

The main objective of the simulation process is the learning that the agent can ac-quire from this process. This means that the agent recognizes the configurations of his environment in which he loses the game. It is necessary that these configurations be stored for future analysis (using the weka system [8]). The figure 3 shows the data flow of the game connects four, you can see the inclusion of the simulation process within the system.

9. FINDINGS

Table 1 shows the number of games that the intelligent agent has won after playing 100 times the game of connect four vs. a human user, it can be seen that in the first row, the data correspond to games without the agent making use of the simulation process. Here the difference of games won between the user and the agent is only two sets. In this version, there are different patterns that the agent is not able to recognize, which can be recognized by a user after at least 4 games.

The second row corresponds to the games in which the agent makes use of the simulation process; the difference between the winning games is a little greater with respect to the version without simulation. Although the simulation process allowed the agent to avoid certain patterns that led him to lose, it is not enough since there are some that are not yet observed by the agent, these patterns were more difficult to recognize by the user, since it took at least 10 games to find one.

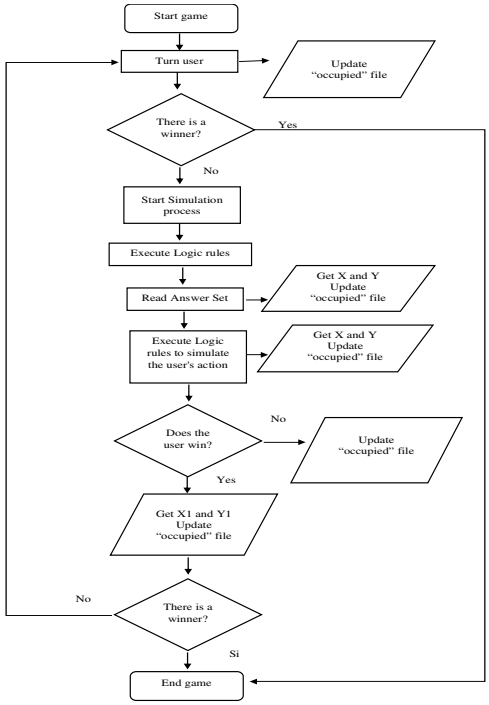

Figure 5. Game Flow Diagram Connect Four

Table 1. Intelligent Agent Records

Version	Number of games	Matches won by Intelligent Agent	Matches won by User	Tied game
Without Simulation Process	200	162	35	3
With Simulation Process	200	194	5	1

10. CONCLUSIONS AND FUTURE WORK

Papers Connect four game is based on logic strategies, the implementation of an intelligent agent as an opponent is a task that involves the modeling of a logical rules system that can describe certain strategies for the agent to execute the pertinent action to in order to achieve your goal: win the game.

When the intelligent agent was implemented solely based on the logical rules system, it showed a competitive behavior at a medium level, executing the basic actions of winning, blocking and building in an accurate way; however, after many tests we noticed that it was necessary to implement more rules through of case based reasoning technique. The CBR technique is based on a set of problems already played. Given a new problem, the most similar problem of the set of problems played is looked for the adaptation of the solution of the most similar problem is returned (adaptation). If the new case or problem solved is correct and representative (revision stage), it is learned (learning stage).

When the simulation process was implemented, the agent's response was raised to a higher level than when he only used the logical rules, now he can recognize when one of his actions generates a configuration contrary to his objective and then changes the response obtained from the logical

rules system. However, the simulation process does not generate an agent capable of learning, it is necessary to store certain items that are identified as patterns that lead the agent to lose the game.

Once these patterns are identified and, stored then it is necessary to perform an analysis and stored, then it is necessary to perform an analysis and processing by using software for knowledge analysis that will give us an answer if it is necessary to generate more rules in order for the agent to acquire learning.

With respect to case based reasoning technique, much work must be done to be able to incorporate the capabilities of agent learning. The case based reasoning cycle possess the capabilities to learn from past and present experiences, thus case based reasoning is a suitable technique appropriate for dealing with the complex decision making [11].

In adaptation, we need to incorporate formal algorithms that support the way in which similarities are sought, as well as the way in which new rules are generated from previous game. In addition, it is necessary to develop efficient algorithms in the way in which formal learning is generated, in order to be able to generalize it and be able to apply it to various problems. Thus, the main objective of this proposal is to define a methodology for the generalization of this game to "Connect k", an interesting problem to analyse.

ACKNOWLEDGEMENT

The authors wish to thank Conacyt for their support of this research. Likewise, we would like to thank the support received from the academic group of combinatorial algorithms and learning. Furthermore, the first and third authors would like to thank the support received from PRODEP. Finally, the second, fourth and fifth authors wish to thank the support received from the faculty of architecture.

REFERENCES

[1] M. Gelfond and V. Lifschitz: The Stable Model Semantics for Logic Programming - Proceedings of the Fifth International Conference on Logic Programming (ICLP), pages 1070-1080, 1988.

[2] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Christoph Koch, Cristinel Mateis, Simona Perri, Francesco Scarcello. "The DLV System for Knowledge Representation and Reasoning", October 2002.

[3] Brouwer L. E. J. (trans. by Arnold Dresden). "Intuitionism and Formalism". Bull. Amer. Math. Soc. 20 (2): pp.:81–96. doi:10.1090/s0002-9904-1913-02440. MR 1559427, 1913.

[4] https://en.wikipedia.org/wiki/Connect_Four.

[5] Number of legal 7 X 6 Connect-Four positions after n plies, Online Encyclopedia of Integer Sequences, 2003.

[6] https://www.aboutespanol.com/conecta-cuatro-2077685.

[7] Dlv System: http://www.dlvsystem.com/

[8] Weka 3: Data Mining Software in Java. https://www.cs.waikato.ac.nz/ml/weka/

[9] Zhao C, Zhao X, Tan Z, Yan X. Research on multiagent system model of diesel engine fault diagnosis by case-based reasoning. First International Conference on Innovative Computing, Information and Control 2006; 386-389

[10] Bokolo Anthony Jnr, Mazlina Abdul Majid, Awanis Romli. Application of Intelligent Agents and Case Based Reasoning.

[11] Leite A, Girardi A. A Case-Based Reasoning Architecture of an Hybrid Software Agent. IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) 2014.

[12] Niemelä, Ilkka & Simons, Patrik & Syrjänen, Tommi. (2000). Smodels: A System for Answer Set Programming. CoRR. cs.AI/0003033.

[13] Yuliya Lierler. SAT-Based Answer Set Programming. dissertation2010. Department of Computer Sciences, The University of Texas at Austin. http://www.cs.utexas.edu/users/ai-lab/?lierler:dissertation2010.

[14] Fangzhen Lin and Yuting Zhao. ASSAT: Computing Answer Sets of A Logic Program By SAT Solvers. AAAI-02 Proceedings, pp. 112-117. 2002.

[15] Berenji, H.R. & Wang, Yan. (2006). Case-Based Reasoning for Fault Diagnosis and Prognosis. 1316 - 1321. 10.1109/FUZZY.2006.1681880.

[16] Wang, Yi and Jiang, Ping and Luo, Yao Hui and Xie, Yan Qun. The Intelligent Fault Diagnosis of Construction Machinery Based on Multi-Agent. Measuring Technology and Mechatronics Automation, Vol. 48. Pp. 1383-1388. Series Applied Mechanics and Materials. Ed. Trans Tech Publications. 2011.

[17] Wikipedia. Connect Four, https://es.wikipedia.org/wiki/Conecta_4

[18] Wladimir Stalski. Enhancement of Cognitive Abilities of an Agent-Robot on the Basis of Image Recognition and Sound Perception. The International Journal of Artificial Intelligence & Applications, Vol. 9, No. 6. 2018.

[19] Mohammad Anisi and Morteza Analoui. Multinomial Agent's Trust Modeling Using Entropy of the Dirichlet Distribution. The International Journal of Artificial Intelligence & Applications, Vol. 2, No. 3. 2011.