pd.tvz.hr
POLYTECHNIC & DESIGN Vol. 7, No. 2, 2019.
DOI: 10.19279/TVZ.PD.2019-7-2-15

ARTIFICIAL INTELLIGENCE IN COMPUTER GAMES

UMJETNA INTELIGENCIJA U RAČUNALNIM IGRAMA

Renata Kovačević, Ivan Cesar, Davor Cafuta

Tehničko veleučilište u Zagrebu, Vrbik 8, Zagreb, Hrvatska

Abstract

Today, the highly developed and competitive computer games industry needs to make better and better computer games and beat the competition. In order to keep the players entertained with computer games, manufacturers use a variety of techniques to make games interesting and challenging. This is largely aided by research in the field of artificial intelligence that is extremely well suited for computer games. Games need to be made as complex and unpredictable as possible to provide as much fun as possible. This article explores and gives an overview of all the most popular techniques that can be applied.

Keywords: *AI, computer games, FSM, MCST, Neural networks, Genetic algorithms*

Sažetak

Danas, visoko razvijena i konkurentna industrija računalnih igara mora proizvoditi sve bolje računalne igre kako bi bila bolja od konkurencije. Kako bi igrače nagnali na što dulje sudjelovanje u igri, proizvođači koriste razne tehnike kako bi one bile zanimljive i izazovne. Ovome u velikoj mjeri pomaže istraživanje u području umjetne inteligencije koja je izuzetno pogodna za razvoj računalnih igara. Igre moraju biti što je više moguće složene i nepredvidljive kako bi pružile igraču zabavu. Ovaj članak istražuje i daje pregled svih najpopularnijih tehnika koje se mogu primijeniti u ovom području.

Ključne riječi: *AI, računalne igre, FSM, MCST, neuronske mreže, genetski algoritmi*

1. Introduction

1. *Uvod*

The definition of artificial intelligence is the ability to learn or understand or to deal with new or trying situations[1]. This definition means that any device programmed to cope with a certain situation can be seen as intelligent. Artificial intelligence is the creation of computer programs that emulate acting and thinking like a human, as well as acting and thinking rationally[2].

Generally speaking, most of the time, AI in computer games refers to path finding and emulating the behavior of other players in the game. These are often referred to as non-player characters (NPCs). With most rules, conditions and actions being prewritten, it often appears more “artificial” then “intelligent”. The concept behind game AI is decision making. To be able to decide some action, the AI needs to have a set of decisions and a set of actions based on those decisions. The first games that started using artificial intelligence were Galaxian (1979) and Pacman (1980). Galaxian used AI to maneuver the enemies. The enemies had complex movements and were able to move out of formation. In Pacman, different movements were added to different types of enemies and it gave a new meaning to labyrinth games[3]. Over the years, AI in games became increasingly complex and today almost no computer game can be made without it.

This paper explores all the most popular techniques of implementing artificial intelligence in computer games and gives an overview of each of them. Every technique is described and a review of their advantages is given. The main goal of this paper is to assist in the selection of a particular technique with regards to the game genre being developed.

2. Finite state machine
2. *Stroj konačnih stanja*

The finite state machines are one the most common AI techniques used in a vast variety of computer games. A finite state machine is a device, or model of a device, which has a finite number of states it can be in at any given time and can operate on input to either make transitions from one state to another or to cause an output or action to take place. A finite state machine can only be in one state at any given moment in time[4]. The general idea of a finite state machine is to disassemble the behavior of an object into easily manageable parts or "states". Every state has its own conditions for each transition into another state.

As an example, lighting can be taken. Any light can have only two states: on and off. The light can be in only one of the state at a moment. It can't be on and off at the same time. By flipping the switch one actually initializes the transition between the two states. Therefore, when the light is off, by flipping the switch the state transitions from off to on. Then, the light is in the state of on. By flipping the switch again, the state makes a transition again, this time from on to off. This is probably the simplest example of a finite state machine. One a bit more complex example can be the life of a cat. It can be said that cats have only four states: relax, sleep, search and eat. When the cat wakes up from sleeping it relaxes. It does this until it gets hungry, then it searches for food until it finds it. When it finds food it eats. If the cat eats just enough food, it transitions back to relaxing. If the cat overeats it goes to sleep. Also, while relaxing it can become tired of relaxing and go to sleep.

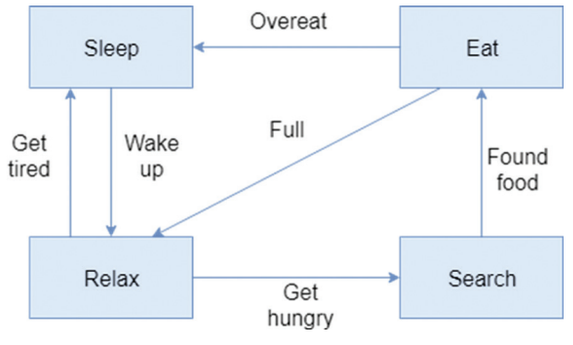

Figure 1 *Finite state machine of the life of a cat*

Slika 1 *Stroj s konačnim brojem stanja iz života mačaka*

Finite state machines can be implemented in several ways. The most straightforward way can be made using if-then statements. Although this approach is reasonable with this simple example, things can get complicated very fast with any larger finite state machine. More states give more programming code, more if-then statements and more case statements. This makes the programming code difficult to read or maintain, and with adding more states debugging becomes an impossible mission. Whilst using this type of implementation, one has to keep in mind that it is not a good option if the game logic is going to be extended.

Another way to implement a finite state machine can be through finite state tables. This method of implementing finite state machines gives a better overview of all the states, transitions and conditions. Furthermore, adding new states, changing and deleting existing ones is more flexible and simplified. It is possible to use the states in the finite state tables in the same way as in the previous example, or it is possible use them in a more complex programming code. This method provides a clean and flexible architecture of a finite state implementation.

Table 1. *Finite state table*

Tablica 1. *Tablica stanja*

Current State	Condition	State Transition
Sleep	Wake up	Relax
Relax	Get tired	Sleep
Relax	Get hungry	Search
Search	Found food	Eat
Eat	Full	Relax
Eat	Overeat	Sleep

Along with the aforementioned methods, implementation can also be embedded inside the classes and programming code. This architecture is known as the state design pattern. This programming pattern provides an elegant simplification of states within the programming code itself.

Within this pattern the Cat class contains data related to the current state that it is in at the moment. The ChangeState method is called to facilitate the transition from one state to another when the conditions are met.

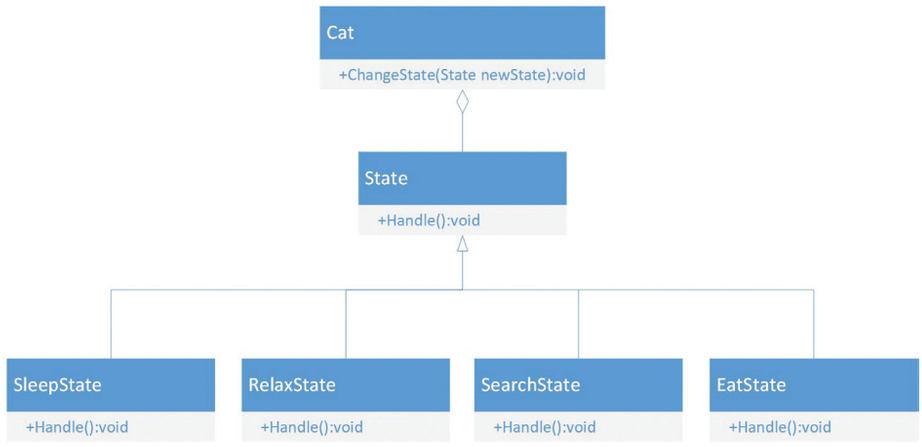

Figure 2 *State design pattern*

Slika 2 *Obrazac dizajna stanja*

These conditions and the transitional logic are implemented within this method. The State class is most commonly an abstract class. It will never be instantiated, but it will be inherited by all possible states needed in the finite state machine. Each state is implemented in the form of a separate class that inherits the base class State. This allows each state to have similar behavior implementation. So, within the inherited Handle method, one can implement the logic associated with that particular state. This pattern allows full encapsulation and the code is easily extended with either states or transition logic. In addition to the states of a character in the game, this design pattern can also be used to control the main game flow components like the menu, save, pause, etc.

Although the use of a finite state machine gives a good overview of possible actions (states) of non-player characters, it also has its drawbacks. One of the drawbacks is its predictability. Because all of the states are pre-programmed they become repetitive and the player can lose interest. In order to have a more complex non-player character it needs to have a large number of possible states which requires a lot of programming and effort.

3. Behavior trees

3. *Stabla ponašanja*

Behavior tree is an AI system that works in a very similar way to a finite state machine. Behavior trees are made up of finite state machines that work in a hierarchical system. Instead of states, the behavior tree technique has behaviors, and instead of state functions, there are various finite state machines.

3.1. The MCST algorithm

3.1. *MCST algoritam*

Monte Carlo Tree Search (MCTS) is a method for making optimal decisions in artificial intelligence problems. These problems usually include movement planning in combinatorial games, but can be applied to any game of finite length[5]. These types of games can be Chess, Go, or any other similar board game [6].

The MCTS is based on the Monte Carlo method. This method is the generation of random objects or processes in order to solve problems that are deterministic in nature. It creates random samples and events, repeating the experiment until the problem is solved. They are often used in physical and mathematical problems[7].

In their work, Chaslot et al. give an example how the MCST algorithm works. As shown in Image 3, the basic MCTS algorithm has four steps that are repeated:

1. Selection – the search starts at the root node with the optimal child nodes selected. This process is performed until a leaf node is reached. This search will be performed by a selection function. Such function can be the finite-horizon Markov decision process (MDP) which is noted as[9]:

$$\frac{w_i}{n_i} + c\sqrt{\frac{\ln t}{n_i}}$$

Where:

w_i – the number of wins after the i-th move

n_i – the number of simulations after the i-th move

c – exploration parameter, needs to be tuned experimentally

t – total number of simulations, equal to the sum of all n_i

2. Expansion – if the chosen leaf node is not a terminal node, a new (child) node will be created and selected. Generally, the tree is expanded by one node for each time the game is simulated.

3. Simulation – simulations on the node created in the last step are played out. The game is played out until the end with the moves chosen randomly.

4. Backpropagation – the current move sequence (the root node) is updated with the simulated result. On each node the number of visits should be incremented. Each node should contain two values: the estimated value based on the simulated results and the number of times it has been visited.

This process is repeated until the allocated time runs out.
An example of the MCTS algorithm can be the game Civilization. In this game, players compete to develop cities. The AI cannot be pre-programmed for every move, so it has to evaluate the possible next moves and identify the best one. In this game three actions are available: defend, build technology and attack. The algorithm creates all the possible actions, calculates the payback for each action, identifies the best action and takes it[10].

MCTS has advantages over the traditional methods. One of these advantages is the fact that the algorithm does not have to have any knowledge about the game. The only thing it should be aware of are the legal moves and at what condition does the game end, enabling it to be efficiently reused. When the algorithm is started, an asymmetric tree is drawn. The more interesting nodes are visited more often, and the search time is lowered with the focus on the relevant part of the tree. This makes the algorithm suitable for games with a large combination of possible moves.

However, the MCTS has also its drawbacks. Due to the large number of combinations of possible moves, the time limit may stop the algorithm before it has calculated the next move. For the algorithm to come up with a good solution, sometimes it needs to explore large number of iterations. Therefore, it could be concluded that its biggest drawback is time.

In order to correct these shortcomings, many solutions have been proposed. Most of these solutions suggest introducing game rules into the algorithm itself. This reduces its reusability, but the algorithm will skip the implausible moves which will reduce the number of iterations[11,12].

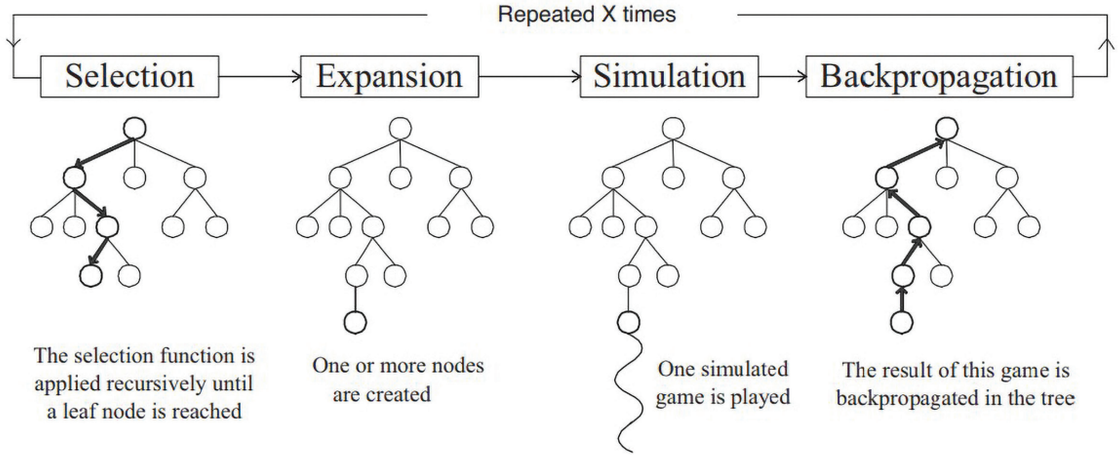

Figure 3 *Outline of a MCST algorithm[8]*

Slika 3 *Pregled CST algoritma[8]*

4. Fuzzy logic
4. *Neizrazita logika*

Fuzzy logic expands conventional logic so that it can handle the concept of partial-truth values between the boolean contrast of true and false[13]. The fuzzy logic reasoning system has the following components: variables, rules and an inference engine. During the process of fuzzification variables get fuzzified by using fuzzy set history and selecting a set of functions on their possible range of values. By using fuzzified values computers can interpret linguistic rules and are able to produce certain outputs. These outputs can remain fuzzified or can be de-fuzzified to provide a crisp value.

Basically, variables can be the input given to the fuzzy inference engine. Firstly, they are fuzzified. In the engine, a set of fuzzy rules are implemented in order to apply certain rules. These rules are usually implemented using the if-then statement logic and boolean operators: AND, OR, NOT. With the help of these statements and operators the fuzzy inference engine "concludes" what the output should be. This output is in the form of a fuzzy value. These values get de-fuzzified to provide a crisp value[14].

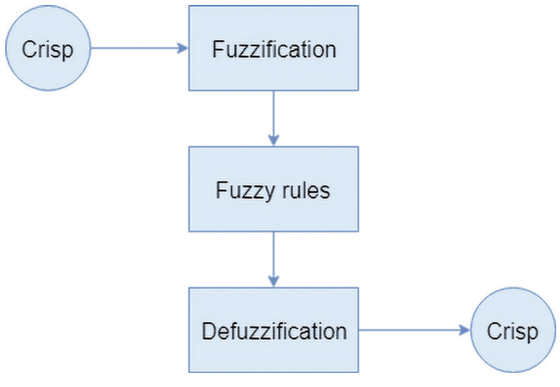

Figure 4 *Fuzzy rule-based inference*

Slika 4 *Zaključak na temelju neizrazitih pravila*

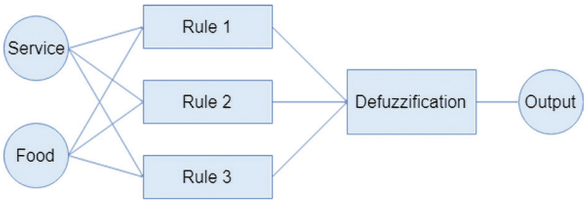

Figure 5 *Fuzzy logic example*

Slika 5 *Primjer neizrazite logike*

Fuzzy logic is simple to implement within a game due to its simple implementation. This simplicity is based on its language-like nature. With fuzzy logic one can implement rules like:

1. IF the service is poor OR the food doesn't taste good THEN don't leave a tip.
2. IF the service is good THEN give an average tip.
3. IF the service if good OR the food is delicious THEN leave a generous tip.

As can be seen in Image 5, the variables are food and service. These variables are fuzzified and given to the fuzzy inference engine which contains the three defined rules. The results are then de-fuzzified and given as an output as a crisp number.

Although the complexity of fuzzy logic does not make it a frequent selection in AI implementation in games, some popular games do use it. One of these games is the Sims - a simulation of human life. The player controls a family of one to eight members. Each member has its own needs that have to be satisfied. The AI in The Sims is specific due to its ability to interact with its environment to meet the sims needs[13].

5. Academic artificial intelligence
5. *Akademska umjetna inteligencija*

There is a distinction between academic artificial intelligence and the artificial intelligence that is used in computer games. Although academic artificial intelligence can be implemented into computer games, because of its complexity, resource and performance constraints usually it is not. The academic AI tends to find the optimal solution regardless of the hardware or time limitations. The methods described beforehand give the developer control over the artificial intelligence in non-player characters. With the use of academic artificial intelligence, such as neural networks and genetic algorithms, reactions of non-player characters in certain situations may be quite unpredictable[4].
Academic artificial intelligence can be divided into strong AI and weak AI. Strong AI handles the problems of how to create a system that can mimic human behavior.

It should understand itself enough to be able to improve itself. Weak AI, on the other hand, handle real world problems and techniques how to solve them. It cannot modify itself in order to improve. It handles only the problem for which it is created[15].

Typically, when advanced game AI is implemented in games, the game is created around the AI and it is not just a part of the game.

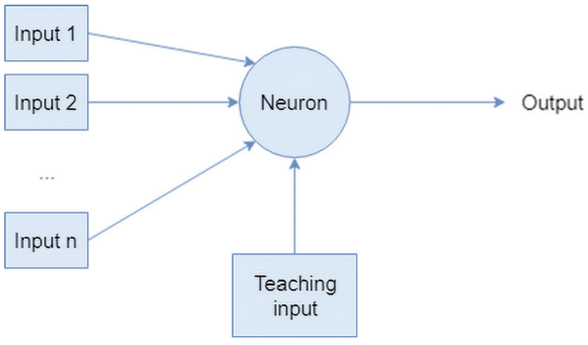

Figure 6 *A simple neuron*

Slika 6 *Jednostavan neuron*

5.1. Neural networks

5.1. *Neuronske mreže*

Neural networks are inspired by the biological nervous systems. They process information in a similar way the brain would. This system is composed of a large number of neurons (elements) which are connected and work as whole. Their key feature is their ability to learn on examples. Each neural network should be specifically designed to solve a certain problem. The way the connections between the neurons are made are always specific to the task they are supposed to do. As seen in Image 6, every neuron has its set of input data and teaching data. When this is combined, it gives out the corresponding output[16].

When combined, neurons can together form a neural network with 3 basic layers: input layer, hidden layer, output layer. On the input layer, the raw data is fed into the network. This is the layer in which data, that should be analyzed and processed, is given to the neural network. The hidden layer is where the data will be processed.

Each neuron and the weights on the connections determine how the "flow" is going to go. From the hidden layer, data is given to the output layer based on the weights on the connections between these two layers. In this simple architecture, in the hidden layer the neurons are free to construct their own representation of the input. The weights on the connections determine which neuron will be active[16,17].

Although the usage of advanced game AI is more of an exception, some games do implement them. One of those games is Black & White where neural networks are used to give "life" to a non-player character. In this game the player is a god who is free to do whatever he pleases. He rules the people in the game. At one point in the game the player is given a creature that, in the beginning behaves as a child. Because of the use of neural networks, the player can "teach" this creature and raise it like one would a child. By praising and scolding it, the creatures' personality develops in the way the player wants it to[18].

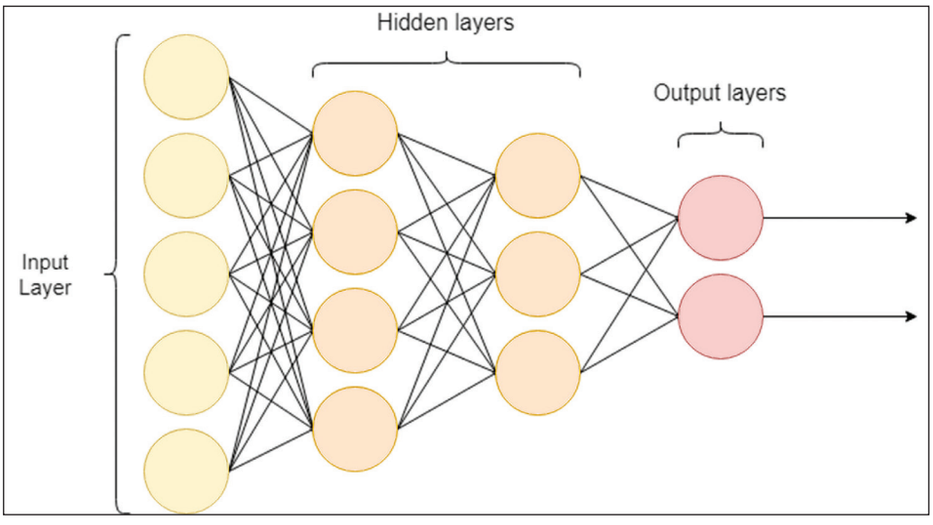

Figure 7 *Architecture of a neural network*

Slika 7 *Arhitektura neuronske mreže*

5.2. Genetic algorithms
5.2. *Genetski algoritmi*

Genetic algorithms mimic the main Darwinian principle – survival of the fittest. This is also the main principle of evolution. The individual with better survival capabilities will be the one who will survive and transfer its genes on to new generations. They solve optimization problems using historical information in order to direct its search into a region in which a better result can be given[19]. Genetic algorithms have the following components: population, chromosomes, gene and allele. The population represents a subset of all the possible solution to a certain problem. A chromosome is one solution to the problem. A gene represents one element position of a chromosome, and an allele is the value the gene takes for a particular chromosome.

Figure 8 *Genetic algorithm components*
Slika 8 *Komponente genetskog algoritma*

This algorithm starts with an initial population. It can be generated randomly or determined and added manually. With the use of a Fitness function calculation a suitability of the solution is produced. Within the crossover, new offspring are generated based on the "parents" (the current chromosomes). Mutation produces random modifications to the current chromosomes. The new offspring replace the existing chromosomes within the population using "the stronger wins" algorithm. This process is repeated until certain criteria is reached. In the end, the process returns the best solution to the given problem. This method is suitable when creating in-game scenarios that include pursuit or evasion and, in general, routing through the game scene. One such example would be a tower defense game where spatial reasoning is needed[20].

6. Conclusion
6. *Zaključak*

Considering the fact that every game is unique in its own way, not every solution can be applied globally to all games. Although game genres are different, they can use the same AI solutions for their problems. A different way of creating AI can be used in many ways and in multiple genres of games, provided that each version is customized for that particular solution.

The Monte Carlo tree search algorithms best works with turn-based strategy games that have a finite length.

Because of their simplicity and ability to keep control over the game flow, finite state machines are often the first choice of implementation in all of the mentioned genres. The rules can be added, combined and controlled quite easily which simplifies the implementation. Also, because they do not need a lot of resources to be executed, other elements within the game can run smoothly. Fuzzy logic is a bit pricier and complicated to implement so it is not the first choice with developers.

It works well with certain genres such as adventure games, Real-Time Strategy Games and First-Person Shooter/Third-Person Shooter games. Neural networks and genetic algorithms are quite pricey in terms of resources needed to be executed. Also, their implementation can be complicated and the behavior of non-player characters chaotic and unpredictable. This limits their use and most games that implemented are built around the AI and it is not just a component of the game.

7. REFERENCE
7. *REFERENCES*

[1.] Intelligence | Definition of Intelligence by Merriam-Webster, https://www.merriam-webster.com/dictionary/intelligence.
[2.] S. Russel, P. Norvig, Artificial Intelligence: A Modern Approach, 3rd ed., Prentice Hall Press Upper Saddle River, NJ, USA ©2009, 2010. http://aima.cs.berkeley.edu/, ISBN: 978-0-13-604259-4.
[3.] D. Charles, C. Fyfe, D. Livingstone, S.

McGlinchey, eds., Biologically Inspired Artificial Intelligence for Computer Games, IGI Global, 2011. http://services.igi-global.com/resolvedoi/resolve.aspx?doi=10.4018/978-1-59140-646-4, DOI: 10.4018/978-1-59140-646-4, ISBN: 9781591406464.

[4.] M. Buckland, Programming Game AI by Example, 1526. http://index-of.co.uk/Game-Development/Programming/Programming Game AI by Example.pdf, ISBN: 1-55622-078-2.

[5.] K. Bayer, S. Koch, R. Klein, Monte Carlo Tree Search, WiSt - Wirtschaftswissenschaftliches Stud. 47 (2018) 11–18. http://mcts.ai/about/index.html, DOI: 10.15358/0340-1650-2018-12-11, ISSN: 0340-1650.

[6.] Y. Bjornsson, H. Finnsson, CadiaPlayer: A Simulation-Based General Game Player, IEEE Trans. Comput. Intell. AI Games. 1 (2009) 4–15. http://ieeexplore.ieee.org/document/4804731/, DOI: 10.1109/TCIAIG.2009.2018702, ISSN: 1943-068X.

[7.] D.P. Kroese, T. Brereton, T. Taimre, Z.I. Botev, Why the Monte Carlo method is so important today, Wiley Interdiscip. Rev. Comput. Stat. 6 (2014) 386–392. http://doi.wiley.com/10.1002/wics.1314, DOI: 10.1002/wics.1314, ISSN: 19390068.

[8.] G. Chaslot, S. Bakkes, I. Szitai, P. Spronck, Monte-carlo tree search: A New Framework for Game AI, in: Belgian/Netherlands Artif. Intell. Conf., 2008: pp. 389–390. https://www.aaai.org/Papers/AIIDE/2008/AIIDE08-036.pdf, ISSN: 15687805.

[9.] H.S. Chang, M.C. Fu, J. Hu, S.I. Marcus, An Adaptive Sampling Algorithm for Solving Markov Decision Processes, Oper. Res. 53 (2005) 126–139. http://pubsonline.informs.org/doi/abs/10.1287/opre.1040.0145, DOI: 10.1287/opre.1040.0145, ISSN: 0030-364X.

[10.] L. Harbing, AI in Video Games: Toward a More Intelligent Game, SINT. (2017). http://sitn.hms.harvard.edu/flash/2017/ai-video-games-toward-intelligent-game/.

[11.] R. Lorentz, Using evaluation functions in Monte-Carlo Tree Search, Theor. Comput. Sci. 644 (2016) 106–113. https://doi.org/10.1016/j.tcs.2016.06.026, DOI: 10.1016/j.tcs.2016.06.026, ISSN: 03043975.

[12.] C.H. Hsueh, I.C. Wu, W.J. Tseng, S.J. Yen, J.C. Chen, An analysis for strength improvement of an MCTS-based program playing Chinese dark chess, Theor. Comput. Sci. 644 (2016) 63–75. https://doi.org/10.1016/j.tcs.2016.06.025, DOI: 10.1016/j.tcs.2016.06.025, ISSN: 03043975.

[13.] M. Pirovano, I. Elettronica, P. Milano, The use of Fuzzy Logic for Artificial Intelligence in Games The current state of Game AI, (2012). http://www.michelepirovano.com/pdf/fuzzy_ai_in_games.pdf.

[14.] U. Kose, Developing a Fuzzy Logic Based Game System, Comput. Technol. Appl. 3 (2012) 510–517. https://www.academia.edu/1919779/Developing_a_fuzzy_logic_based_game_system.

[15.] K. Chethan, Artificial Intelligence: Definition, Types, Examples, Technologies, Medium. (2018). https://medium.com/@chethankumargn/artificial-intelligence-definition-types-examples-technologies-962ea75c7b9b.

[16.] I. Goodfellow, Y. Bengio, A. Courville, Deep Learning, MIT Press, 2016. http://www.deeplearningbook.org, ISBN: 978-0262035613.

[17.] C. Stergious, D. Siganos, Neural Networks, https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html.

[18.] J. Wexler, Artificial Intelligence in Games: A look at the smarts behind Lionhead Studio's "Black and White" and where it can and will go in the future, 2002. https://www.cs.rochester.edu/~brown/242/assts/termprojs/games.pdf.

[19.] C. García-Martínez, F.J. Rodriguez, M. Lozano, Genetic algorithms, in: Handb. Heuristics, Academic Press, 2018: pp. 431–464. https://www-sciencedirect-com.ezproxy.lib.ukm.si/science/article/pii/B9780124095458000054, DOI: 10.1007/978-3-319-07124-4_28, ISBN: 9783319071244.

[20.] P. Huo, S.C.K. Shiu, H. Wang, B. Niu, Application and comparison of particle swarm optimization and genetic algorithm in strategy defense game, in: 5th Int. Conf. Nat. Comput. ICNC 2009, IEEE, 2009: pp. 387–392. http://ieeexplore.ieee.org/document/5364640/, DOI: 10.1109/ICNC.2009.552, ISBN: 9780769537368.

AUTORI · *AUTHORS*

Renata Kovačević - nepromjenjena biografija nalazi se u časopisu Polytechnic & Design Vol. 5, No. 1, 2017.

Korespondencija
renata.kovacevic@tvz.hr

Ivan Cesar
Diplomirao 2011. kao magistar inženjer računarstva na Fakultetu Elektrotehnike i Računarstva u Zagrebu, 2012. izabran u suradničko zvanje Asistenta, 2018. u zvanje predavača na Tehničkom Veleučilištu u Zagrebu. Objavio je 2 znanstvena rada (Towards a Method to Retrieving Business Process Model from Source Code // 9th Iberian Conference on Information Systems and Technologies, CISTI 2014; Improving the Simple RFID Reader // TISKARSTVO & DIZAJN 2014), osvojio zlatnu nagradu na 38. Hrvatskom salonu inovacija s međunarodnim sudjelovanjem za MEX uređaj za kontrolu pristupa (uz autore D. Cafuta, I. Dodig, T. Kramberger), aktivno sudjeluje na projektima implementacije računalnog vida u kontekstu upravljanja sigurnošću te potpore poslovanju.

Korespondencija
icesar@tvz.hr

Davor Cafuta - nepromjenjena biografija nalazi se u časopisu Polytechnic & Design Vol. 5, No. 1, 2017.

Korespondencija
davor.cafuta@tvz.hr