MACHINE LEARNING ALGORITHM FOR INTELLIGENT BOTS IN MULTIPLAYER VIDEO GAME: A CASE STUDY

*Teodor Ukov, Georgi Tsochev**

Technical University of Sofia, Sofia
Bulgaria

* Corresponding Author, e-mail: gtsochev@tu-sofia.bg

Abstract: This paper describes the results of a developed online machine learning method implemented in a real-time multiplayer game called Hram Light. The goal was to create a program which produces several ways of playing for a bot, which would become a virtual trainer for players. The solutions are required to vary in success rate in order to achieve progress in player improvement. The input of the method is a data model of parameters which describe a player's way of playing in a game situation. The process of learning takes place in the online world and produces a variety of solutions via a genetic algorithm.

Key words: genetic algorithm, Hram Light, machine learning method.

1. INTRODUCTION

The goal of this article is to provide new ideas for an application of genetic algorithms which could be used in teaching of bots intended for a real-time online multiplayer game. Together with this stands the strive for establishing a solid understanding and definition of the metaheuristic terms and more precisely the genetic algorithms. A nine-month university work, part of the program student's hub of Technical University of Sofia brought an application of Genetic Algorithm (GA) in an online game environment. A special mode for the game client was developed in order to reproduce game situations. This way the mode can be used for training the bot to play in different ways. The future target of application of this work is to "teach a teacher" for it to be a digital trainer bot (DTB) for players.

1.1. Work done and history

Since a year Hram Light has been established to be a university project producing studies and articles in the sphere of Internet Systems and Digital Training. It started as a project for developing a game engine on C++ for online multiplayer games. Effort was put into developing a reusable Application Programming Interface

(API) that is serving the client-server system (C-SS). This brought a coding convention norm and methods for establishing readable and reusable classes. A C++ library was made called Hram Utility, which can be used to describe an abstract mathematical 2D world and help the establishment of a solid communication process. It plays an important role for the reusability of the API. Used in both programming projects, the classes of Hram Utility solidified the development process and helped find different methods for the TCP data transmissioning, which have been applied in the C-SS.

1.2. Main project goals and ideas

There are many studies that show that video games can improve reaction-time and decision making [1, 2]. An action game requires its players to react quickly and notice important game events. Studies have been made on what are the effects on the reaction time comparing gamers to non-gamers. The results were positive - playing action games decreases reaction speed and improves contrast recognition in the visual processing network [3]. The game Hram Light is a 2D action game that aims at becoming a digital trainer, which improves the players' personal qualities. Since a year Hram Light has been established as a university project producing studies and articles which present different methods for improving the personality's fluid reasoning, reflexes and teamwork. Two experiment methods have been designed and planned to be applied on real people, in order to study the effects of digital therapy. One of them includes two teams of players competing against each other, whilst the other represents a digital training process, where one player is playing with one bot. The last requires a specifically designed bot - DTB, which needs to have several, growing in success rate (SR), ways of playing. The SR is the positive game results according to a specific way of playing. It can be most precisely defined by the fitness function of the GA.

There are three goals that describe the overall general purpose of the project Hram Light. They are applied in all aspects of the project including cognitive psychology research, user interface design, exercises, studies and programming work. They are:

1) Develop the personal qualities of the players:

This goal represents the direction of establishing the game as a digital trainer, which contains several modes, each of which is handled in a separate server room.

2) Create a competitive environment for players:

Avoiding the stochastic processes in the gameplay contributes to the competitive characteristic. The idea is that, when players aim to be better than the others, they try to improve themselves.

3) Inspire the player:

The ideas that please one's mind remain in the memory, and therefore the learning process is more efficient [4].

2. DEFINITIONS AND IDEAS

To design and understand the purpose of a GA in a Machine Learning (ML) system a specific definition is required for the terms that describe the observed phenomena and operations. There are three important definitions, described in each subsection below. They have been created in the terms of this work but can be applied in other projects as a way for easier decision making when designing and implementing ML with GA [5, 6].

2.1. Artificial intelligence definition and purpose

A definition of Artificial Intelligence (AI) was made and represented in a real online world with bots. As a field in Informatics, AI defines the so-called rational agents which represent an object that acts in an environment via an actuator and observes it with its sensors. They are 'rational' because they make reasonable decisions and are 'agents' because they apply decisions.

In the terms of this work artificial intelligence is a characteristic observed only if the following three elements are available - Environment, Actor and Goal. Without a goal, AI cannot be manifested and achieved. The observation of AI is done when an application of a bot's way of playing is providing better game results. Thinking of AI in this way gives us more freedom in designing and applying machine learning methods. Many decisions can be made which are not reasonable, but they can be filtered by the learning process. This way the target goal of having different SR is achieved. On the other hand, this leads the way towards the development of the game mode for the DTB

According to the project's definition for AI the environment of the AI is the game situation (a term that is going to be explained in the next subsection). The actor is a bot - a fully reproduced player with its client. The decisions that are being made are the 9 directions of movement which are changed according to the movement of the opponent - the teacher. The current goal of the AI depends on the game situation but is globally defined as achieving best results against a specific teacher for a period. This result and its calculation are further defined in section 3 where the fitness function is described.

A future goal is to change the bot SR dynamically - according to what the training player is doing and apply machine learning during the digital training itself. Another idea for the future is to add special functionality for redoing the mistakes that the training person has made. This way the bot is going to be pointing at what the player should improve.

2.2. Defined machine learning terms

A genetic algorithm is a metaheuristic algorithm which is not problem dependent - it can be used for many different optimization problems [7]. In order to implement a GA a genetic individual (GI) definition is required. When it comes to searching for solutions in a game world it is of great importance to define what exactly a solution is in order to choose what the genes from the algorithm are going to represent. In our GA it was defined that one GI is a solution of bot input parameters for a specific game situation. When they are passed to the top-level class – BotWill they achieve a specific way of playing. It is not only required to find the best solution in terms of results like positive game points, but a variety of solutions that differ in SR. That is why the choice of GA is the right metaheuristic approach. Ideas for avoiding the local optima problem are defined in the third section [8].

The "game situation" is also an essential element of the learning process because the GIs might be good for one case but bad for another.

The game situation is described by three factors:

1) Goal of the player (in this case the bot)
2) Area on the map
3) Period of the game time

Each GI is dedicated to one game situation. Currently depending on the two game competitive modes (battle modes) the gaming process may define around 5 game situations. The current implementation of the machine learning process is for the game situation of defending the base (a crystal with health points) from an attacking teacher. The period for the situation is considered as 12 seconds. This is because it has been observed from videos of players that an outcome (one of the players is taken down) in this game situation is produced for around that time.

When it comes to finding a solution for the whole gaming process another term emerged - "individ*"*. It describes the combination of GIs that are for all the situations in a game mode, and which describe the whole solution in a game mode.

An AI model in the field of machine learning is a data structure that contains the data produced from the learning data. In this work the GI data is stored in such a file together with the id of the GI, the generation - the sequential number of the generated population from which the GI is and the situation to which it is dedicated. The data structure of the GI is described in the third section.

The future idea is to make analyses based on the personal characteristics and define player types. Then a functionality is going to be made which would design an individ that gives best results in the learning process for a specific player.

3. DESIGN AND IMPLEMENTATION OF THE LEARNING METHOD

The implemented high level programming classes for the bot and the machine learning method are reusing the high-level classes of the game. This way a bot fully reproduces a game client. They are developed frame-wise - they implement their functionality on each frame of the game loop. The classes that achieve the specific gameplay based on the GI took most of the development work. The design of the class hierarchy and separateness, however, established a robust architecture because adding new genes and their corresponding functionality is now quite easy. This was proven when the last genes - the directions to throw at, were designed and added in less than 1 hour.

A game map specifically designated for the purpose was made and a bot command system developed. Currently there are two modes for the bot state of the client - “free mode” that gives the opportunity to the user to set the number of bots that are going to be connected and “machine learning mode”.

Each Bot class has the following compositional classes:

- Individ: Contains the AI models
- BotMind (BM): Carries out the learning process
    - Observer: Creates world events like (opponent changes direction, throws a ball…)
- BotWill (BW): Carries out the learning process
- The game client classes – WorldState and ClientCore (CC) which carry out the pre-established gaming process.

3.1. Machine learning system

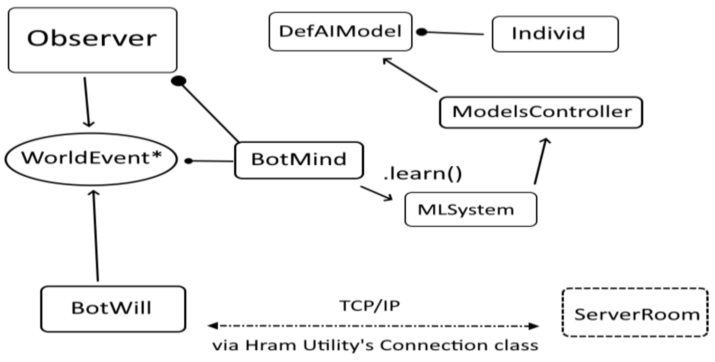

Fig. 1. UML Class Diagram of the classes taking part in the learning process.

The diagram above shows the classes composition and connectedness. For each game frame on each bot the WorldState logic function is applied. After that the Observer generates world events like “I am taking damage”, “Opponent changes direction” and eight more. The collected from each frame events are consumed from either the BM learn method or the BW sync. The damage taking events for example play an important role for calculating the fitness of each GI and so are consumed by the learn method, where the GA is implemented. If there is an event like “Opponent changes direction”, then it is consumed by the BW with its compositional classes for achieving the bot way of playing. They extract the solution parameters from the model data structure and use them to make decisions for direction of movement changing and send TCP packet that corresponds to that decision. The MLSystem (MLS) class belongs to the BotState and it is passed only to the BM learn method. It carries out the GA operations and stores the model’s data structure. It loads the models via the ModelsController which is specifically designed for model files reading. The world events dictate the fitness calculation made in the MLS when the validation (fitness) process for one GI is finished. The code for the common GA operations is implemented in the MLS which provides several easy to configure settings. Section 4 shows one design which was used in the described experiments.

3.2. The GI solution parameters

The combination of the keyboard buttons for movement - W, A, S, D forms 9 directions in the 2D world: none (0) up (1), up-right (2), right (3), down-right (4) down (5), down-left (6), left (7), up-left (8). One of these directions represents one gene in the GA. The term "corresponding directions" emerged - a container of 9 directions the indexes of which correspond to the ordered directions. An example in fig. 2 is shown below.

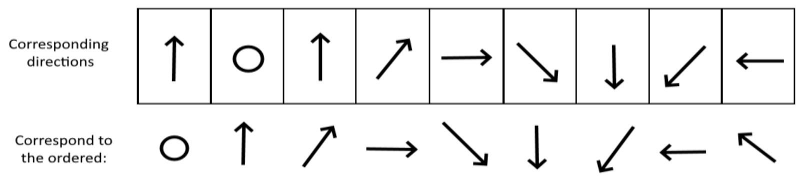

Fig. 2. Corresponding vector - "react to movement" of teacher.

The Corresponding Directions (CD) data structure is stored in the C++ container vector and is accessed by the hram::Dir which corresponds to each of the 0 to 9 indexes in the container. The example shown above is the CD for "react to move directions" which is used by the teacher. The BotMoveAct class, that established the decision making for movement, is triggered when the observer produces a world event of type "Target changes direction". The direction that the target has made is passed to the corresponding vector which returns a corresponding direction. A constant time parameter is defined for which the corresponding direction of movement is going to be persistent. Currently this time is 500ms. The react to movement CD gives the bot the characteristic of dodging the opponent's ball projectiles which after being thrown remain on the ground for 1500ms. They represent balls which have slowing velocity and are generated for a small period (depending on the exact ability between 300ms ~ 1500ms).

The lastly implemented CD was the "directions to throw at". The corresponding direction of that container gives the target of throwing of the projectiles, and it depends on the direction of the opponent. In the case of throwing projectiles (throwables) the directions are calculated with trigonometry functions and are many in count. The projectile that is thrown towards a given direction is being generated for some time, but the opponent towards it is thrown, does not know that direction. They may only guess and change their movement based on where they suspect their opponent is aiming at. That is why the "directions to throw at" genes play an important role in reproducing a real situation where players strive to land their projectiles.

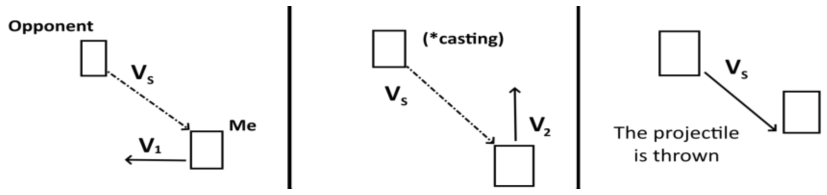

Fig. 3. React to casting scene

The other important CD container is the “directions to react to casting”. The scene is presented in fig. 3 where the strive vector – Vs can be seen. It is named according to the “relative striving” exercise created in the game for personal qualities training. Vs is immutable according to its owner body center - the caster can move while casting. In order to present the perspective on the scene of fig.3, the bot that is using the CD is named ‘Me’. When the opponent starts to cast (to generate) a throwable for a period, the ‘Me’ bot gets the corresponding from the CD container direction to its own movement. The solution parameters for the “directions to react to casting” are establishing the AI characteristic of dodging higher speed projectiles.

There is one more gene container - “dance directions”. The directions in it do not correspond to others but are order-based genes [8]. They are applied one after another in 300ms period and form the specific for the GI dance which as the CD “react to move” can also play a role in the dodging characteristic of the AI. When a react to move direction is made the period for which it is persistent is 500ms. If there is a case when direction has not been changed for more than 500ms then the bot starts its so-called “GI dance”. This does not occur so often in the machine learning process, but more so when the bots are playing against real players.

3.3. The learning method

All the solution parameters containers described above participate in the crossover and mutation. A reset functionality with a fitness period variable was developed in order to establish the GI evaluation process. The so-called both commander - the user that runs the client in bot state can control the bots with the keyboard. Currently there are three operations available:

- Pause bots: all of them remain on the spot where they are.
- Reset bots: all of them are spawned at their beginning point, have their game points reset and their fitness data.
- Save current population: the algorithm can be performed continuously, and the user decides when to save a required generated population.

A key moment for the learning method is that it is performed in a real online world environment where a variety of packet transmission time is observed. Because all the decisions being made are like “connected vessels”, the packet’s travel time difference influences the outcome of the bot's performance. Even having the same solution parameters for the teacher and the evaluated GI, the fitness value may vary for two evaluation processes. This may be referred to the concept drift problem [9] and can be solved with further designing on packet travel time monitoring.

3.4. The algorithm

1) Get initial population:

Model files in the client are read and passed to the system. They have been manually made by the engineer with the strive to have variety. The genes for reacting to an attack on the vertical axis were intentionally set.

2) Evaluate:

Fitness function: fitness = $x-y-3z+t(b-x)/3-f(b-y)/3$, where
Variables: x - damage done; y - damage taken, z - base damage taken, t - takedowns; f - falls,
Constants: b - body energy (151)

3) Select the first 3 best GIs:

After all the population is evaluated, it sorts GIs by fitness and get the first 3.

4) Crossover the first 2 GIs:

Generate 2 and mutate the genes of each of their CD containers with indices 4, 5, 6

5) Mutate the third best

6) Create next population and go to step 2:

Stores the 2 best GIs from the previous population, their 2 children and the third best mutated.

4. EXPERIMENTAL SETTING

This section presents a single experiment – an online session of the method. Solutions which vary in style and success rate of playing have been produced according to the provided input parameters – the GI of the teacher. The zone on the game map is simplified – does not have obstacles in the center and has the two bases vertically aligned. The teacher attacks from bottom to top. A session can be processed for infinity, but useful solutions are available after around 2~3 minutes. This was noticed because after this time most of the genes of the GIs were equalized. A save command can be called at any time without interrupting the session, thus providing solutions with a lower SR. In the case of this experiment the method was applied for around 20 minutes. The same genes pattern emerged in the non-mutated genes of the GIs which shows the best in terms of SR way of playing and an alternative – the mutated genes.

4.1. Experiment

In order to present the results easier has been decided to describe the genes by the numbers from 0 to 8 correspond to the 9 directions clockwise starting from standing still:

0: still, 1: up, 2: up-right, 3: right, 4: down-right, 5: down, 6: down-left, 7: left, 8: up-left.

Table 1. Teacher GI

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dance	*1*	*4*	*1*	*6*	*1*	*4*	*1*	*6*	*1*
React to move	*1*	*1*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
React to throwable	*0*	*5*	*6*	*7*	*8*	*1*	*2*	*3*	*4*
Dirs to throw at	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

Most of the genes of the teacher were randomly designed except the ones that make the bot move upwards. The first directions in the react to move CD for example have the thought of going upwards (1 is up) when the opponent (learning) bot is standing still or is going upwards. The other experiment settings are:

- Fitness time: 10 seconds
- Mutating the downwards genes (4, 5, 6)
- Crossover on 2 children from the 2 best
- Mutating (0, 1, 3, 5, 7) on the third best

Initial population:

Table 2.GI with ID: 0

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dance	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
React to move	*1*	*1*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
React to throwable	*1*	*1*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Dirs to throw at	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

Table 3. GI with ID: 1

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dance	*1*	*1*	*3*	*3*	*5*	*5*	*6*	*7*	*7*
React to move	*5*	*5*	*5*	*5*	*6*	*5*	*4*	*6*	*1*
React to throwable	*0*	*1*	*2*	*5*	*4*	*5*	*6*	*7*	*8*
Dirs to throw at	*1*	*1*	*3*	*3*	*5*	*5*	*6*	*7*	*7*

Table 4. GI with ID: 2

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dance	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*8*
React to move	*5*	*2*	*1*	*3*	*4*	*4*	*1*	*1*	*1*
React to throwable	*1*	*2*	*3*	*2*	*1*	*4*	*5*	*5*	*1*
Dirs to throw at	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*8*

Table 5. GI with ID: 3

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dance	*1*	*1*	*1*	*2*	*8*	*1*	*1*	*2*	*1*
React to move	*5*	*1*	*2*	*5*	*0*	*0*	*1*	*1*	*2*
React to throwable	*0*	*1*	*2*	*5*	*4*	*5*	*6*	*7*	*3*
Dirs to throw at	*1*	*1*	*1*	*2*	*8*	*1*	*1*	*2*	*1*

Table 6. GI with ID: 4

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dance	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
React to move	*5*	*1*	*2*	*5*	*0*	*0*	*1*	*1*	*8*
React to throwable	*0*	*1*	*2*	*5*	*4*	*5*	*6*	*7*	*8*
Dirs to throw at	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*

Some useful GIs that have reached the10th generation:

Table 7. GI with ID: 14

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dance	*0*	*1*	*2*	*3*	*2*	*7*	*4*	*5*	*1*
React to move	*5*	*1*	*2*	*5*	*3*	*6*	*4*	*4*	*2*
React to throwable	*0*	*1*	*2*	*5*	*7*	*2*	*0*	*1*	*3*
Dirs to throw at	*0*	*1*	*2*	*3*	*2*	*7*	*4*	*5*	*1*

Table 8. GI with ID: 15

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dance	*0*	*1*	*2*	*3*	*5*	*1*	*7*	*5*	*1*
React to move	*5*	*1*	*2*	*5*	*6*	*0*	*7*	*4*	*2*
React to throwable	*0*	*1*	*2*	*5*	*1*	*5*	*3*	*1*	*3*
Dirs to throw at	*0*	*1*	*2*	*3*	*5*	*1*	*7*	*5*	*1*

Table 9. GI with ID: 16

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Dance	*3*	*4*	*2*	*6*	*5*	*4*	*7*	*8*	*1*
React to move	*8*	*4*	*2*	*8*	*6*	*3*	*7*	*7*	*2*
React to throwable	*3*	*4*	*2*	*8*	*1*	*8*	*3*	*4*	*3*
Dirs to throw at	*3*	*4*	*2*	*6*	*5*	*4*	*7*	*8*	*1*

4.2. Analysis on results

Even though the evaluation process of this method is long (10 s) diverse and logical ways of playing have been produced. A local optimum was found in indices 0, 1, 2 and 3 (see tables 7 and 8). GIs with IDs 14 and 15 from the 10th generation have emerged as the best solutions to the aggressive teacher – notice the reaction to the attack. When the teacher goes up (1 and 2) they respond by also going up with the 4 of their CD containers. GI with ID 15 is more aggressive – has more down (5) reactions and this helped it to do more damage to the opponent's base after taking it down. Both GIs 15 and 16 managed to take down their opponent in 10 seconds, but 16 emerged as more tactical. Without slaying its opponent, it was steadily pushing it downwards, dodging many of the enemy projectiles. In table 9 you can see the appearance of 8 (up-left) directions which are towards its base according to the location of the clash of the 2 bots. With the strive to produce different SR in terms of damage done, it can be said that GI with ID 16 is a lot valuable, as it does not slay its opponent but defends the area in front of its base – the goal of the AI.

CONCLUSION

An automatically applied method for finding a variety of solutions against a specific way of playing is a step towards the target of a smart digital trainer bot. The produced solutions are now provided in a new game mode where players can test them. In the future the DTB is going to simulate a player who loses concentration, by changing to a GI parameters with lower SR while playing against a human. Research is going to be done in order to see if people notice the change in the bot's SR, which is going to give the next directions for the digital training mode. Soon experiments are going to be done with pre- and post- psychological examination on the players after taking a battle against the DTB. A video on the work done is soon going to be created, where the bots in the game world can be seen.

ACKNOWLEDGMENTS

The described ideas and terminologies have been born from an elaboration in progress dedicated to the project "Developing Artificial Intelligence for a Video Game Trainer for Personal Qualities", approved by the Technical University of Sofia commission according to the program "Students Hub" for supporting youth scientists.

REFERENCES

[1] Jordan, T., Dhamala, M., Video game players have improved decision-making abilities and enhanced brain activities, *Neuroimage: Reports*, vol. 2, no. 3, 2022, 100112, ISSN 2666-9560, https://doi.org/10.1016/j.ynirp.2022.100112.

[2] Hutchinson, C.V., Barrett, D.J.K., Nitka, A. *et al.* Action video game training reduces the Simon Effect. *Psychon Bull Rev* 23, 2016, pp. 587–592. https://doi.org/10.3758/s13423-015-0912-6.

[3] Steven Pace. Cognitive Science. A*ction video games improve goal-directed reaction times, study finds* Available at: https://www.psypost.org/2016/06/action-video-games-improve-goal-directed-reaction-times-study-finds-43507 (visited on 13.09.2022).

[4] Oleynick, V.C., Thrash, T.M., LeFew, M.C., Moldovan, E.G., Kieffaber, P.D. The scientific study of inspiration in the creative process: challenges and opportunities. *Frontiers in Human Neuroscience*, vol. 8, 2014. DOI: 10.3389/fnhum.2014.00436

[5] Kravets, O. Ja.; Aksenov, I.; Rahman, P. A. et al. Algorithmization of image processing for identification of dynamic objects. *International Journal on Information Technologies & Security*, vol. 14, no. 3, 2022, pp. 47-58

[6] Basheleishvili, I, Tsiramua, S., Bardavelidze, A. Algorithmization and realization of the software tool for the software code quality assessment. *International Journal on Information Technologies & Security*, vol 14, no. 2, 2022, pp.27-38.

[7] Dana Bani-Hani. Genetic Algorithm (GA): A Simple and Intuitive Guide, 2020 Available at: https://towardsdatascience.com/genetic-algorithm-a-simple-and-intuitive-guide-51c04cc1f9ed (visited on 08.09.2022).

[8] Rocha, M., Neves, J., Preventing premature convergence to local optima in genetic algorithms via random offspring generation, *Multiple Approaches to Intelligent Systems. IEA/AIE 1999. Lecture Notes in Computer Science*, vol 1611, pp. 127-136. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-48765-4_16

[9] Shahapurkar, A., Rodd, S.F. Efficient Feature Aware Machine Learning Model for Detecting Fraudulent Transaction in Streaming Environment. *International Journal on Information Technologies & Security*, vol. 14, no. 3, 2022, pp.3-14

Information about the authors:

Teodor Ukov – engineer at Technical University, with a master's degree. Working as a Full-Stack Software developer. Fields of research: Internet technologies, software design, game engine and game development, machine learning, artificial intelligence and digital therapy.

Georgi Tsochev – assist. prof. at the Technical University of Sofia and has experience as a system and network administrator. His fields of research are network and information security, artificial intelligence, and e-learning.

Manuscript received on 10 October 2022