STUDIA INFORMATICA
Nr 1 (26) Systems and information technology 2022

Jerzy TCHÓRZEWSKI[1]
ORCID: 0000-0003-2198-7185
Arkadiusz BOLESTA[1]
ORCID: 0000-0003-1719-1109

[1] Siedlce University of Natural Sciences and Humanities
Faculty of Exact and Natural Sciences
Institute of Computer Science
ul. 3 Maja 54, 08-110 Siedlce, Poland

Neural model of the vehicle control system in a racing game. Part 1. Design and its implementation

DOI: 10.34739/si.2022.26.02

Abstract. The publication consist of two parts. Part 1 contains the results of research on the design, learning and implementation of the Perceptron Artificial Neural Network as a model of neural control of car movement on the racetrack. This part 1 presents the results of studies, including review of the methods used in video racing games from the point of view of the selection of a method that can be used in the own research experiment, selection of the Artificial Neural Network architecture, its teaching method and parameters for the intended research experiment, selection of the data measurement method to be used in ANN training, as well as development design of a car game, its implementation and conducting simulation tests. In designing the game of vehicle traffic on the racetrack, among others, Godot Engine game engine and MATLAB and Simulink programming environment. The numerical data (14 input quantities and two output quantities) for ANN training were prepared with the use of semi-automatic measurement of the race track control points. Part 2 shows i.a. the results of the testing and simulation experiments that confirm the correct functioning of both the game and the model of the neural control system. There were also shown, among others, the possibility of continuing research in the field of increasing the flexibility of the racing game, in particular the flexibility of the vehicle traffic control system through the use of other artificial intelligence methods, such as ant algorithms or evolutionary algorithms.

Keywords. Artificial Neural Networks, Godot Engine, MATLAB i Simulink environment, CLion IDE, Video games

1. Introduction

Artificial neural networks as methods of artificial intelligence are used, among others for modeling systems, incl. for modeling vehicle traffic control systems in racing games. Racing games are computer games consisting in racing vehicles controlled by many players with the use of computers most often located on the Internet or with use of a split screen [1, 6].

While racing, players try to get the highest possible place, e.g. in scoring, drive the designated route in the shortest possible time or drive the longest possible route in the allotted time. Players can also race against computer players learned to move on the racetrack at different difficulty levels. Racing games include, among others games: Burnout, Forza Motorsport, Gran Turismo, Need for Speed or Test Drive and a number of other games, as well as simracing computer racing [15, 37].

Racing games also include video games in which the main goal is to get the player from the starting point to the finish line in a certain award-winning order, i.e. first. In the control system, a control system and a control object can be distinguished. The control system in racing games is the player equipped with the ability to drive the vehicle, and the objects of control are, among others car-type vehicles intentionally moved on a racetrack. Due to the basic goal of the racing game, the most important phenomenon is the competition between a player equipped with the ability to control an object, which is a vehicle, with other players equipped with similar capabilities, in particular with a player called a computer opponent equipped with appropriate algorithms enabling the vehicle to be driven automatically on the racetrack.

In computer games, the computer opponent is equipped with various types of vehicle control support, including ready-made solutions for its movement on the racetrack, and usually specific movements depend on the player's decision. The movement of the vehicle on the racetrack is constantly recorded using, inter alia, checkpoint measurement system combined with various shortest path finding algorithms such as the A* algorithm or the Dijsktra algorithm [3, 17].

On the other hand, artificial intelligence methods such as artificial neural networks can be used to control vehicles along the game track. The Artificial Neural Network (ANN) learned to control is a very accurate model of the neural system of vehicle motion along the racetrack. As part of the detailed research, the results of which are presented in [6], in order to obtain a model of the neural system of vehicle motion control in a racing game on a specific race track, a Perceptron Artificial Neural Network of vehicle motion was designed and taught using the MATLAB and Simulink environment [25].

In addition, the C ++ language was used to implement the neural network and a multi-platform integrated C / C ++ programming environment called CLion by JetBrains was used [11]. On the other hand, the environment for conducting the game flow experiment was designed with the use of the Godot Engine [8, 10, 24].

The Perceptron ANN used in the neural model was learned with the use of the backward error propagation method. The numerical data for teaching the Artificial Neural Network was obtained by implementing a function that saves the input values to the file as inputs from the vehicle motion system and the output values as responses obtained in the course of manual vehicle control in the developed programming environment.

The Perceptron ANN has been sufficiently learned and tested in 32 research experiments, resulting in a finally tuned neural model of the vehicle control system in a racing game. The longest running time of the vehicle on the track was 4 minutes and 55 seconds, and the shortest 10.47 seconds. On the other hand, within 5 minutes, the highest number of complete laps of the track was 34, and the smallest - 1 and 5 [6].

2. Review of vehicle control methods in racing games

The vehicle control systems in racing games use, among others, the following methods: shortest path finding algorithms (non-directed BFS and DFS algorithms), Dijkstra algorithm and A* algorithm; dynamic path finding algorithm; Racing Lines algorithm, as well as an algorithm using a system of checkpoints, as well as artificial intelligence methods [2, 5, 7, 11, 18, 35].

In the case of shortest path finding algorithms, non-directed algorithms are often used, namely the Depth-First Search (DFS) algorithm based on the depth search algorithm and the Breadth-First Search (BFS) algorithm based on the breadth search algorithm. The essence of these algorithms is based on the nature of the behavior of the rat looking for a path in the labyrinth, which leads to obtaining a solution at all, not necessarily on the shortest possible way out of the labyrinth. His behaviour boils down to consistently checking all possible paths of the labyrinth until it hits the exit, that is, it accomplishes the set goal [2, 6, 11].

For obvious reasons, such a search for an exit from the maze involves a high computational cost of this type of algorithms and therefore they are not used to drive vehicles in racing games. Both of the mentioned algorithms treat the labyrinth map as a graph of connected points, and the DFS algorithm finds the existing path, but does not guarantee that the found path is the shortest. On the other hand, the BFS algorithm searches the graph one step in all possible

directions (Fig. 1) and if there are paths, then the algorithm finds the shortest path among them, but this algorithm does not always work in specific cases, such as, for example, in the case of a weighted graph [2, 6].

The Dijkstra algorithm was developed by the Dutch computer scientist Edsger Dijkstra, who is remembered primarily for the described algorithm and the examination task concerning the problem of dining philosophers [3, 6]. The algorithm is used to determine the shortest distance from the fixed vertex s to all other vertices in the directed graph, while the graph cannot contain edges with negative weights. Having a given graph with a selected vertex (source), the algorithm finds the distances from the source to all other vertices of the graph. It is easy to modify it so that it searches only for the shortest path to one fixed vertex, stopping the operation when it reaches the target vertex.

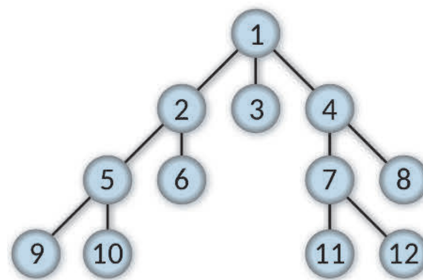

Figure 1. The order of visiting nodes in the BFS algorithm. The markings: consecutive numbers in circles mark subsequent nodes. Source: [2].

On the other hand, the A* algorithm is used to find the shortest path in a weighted graph from any vertex to a vertex that meets a certain condition called the target test [17]. The algorithm is complete and optimal in the sense that it finds a path as long as such a path exists and is at the same time the shortest path. It is mainly used in artificial intelligence tasks to solve specific problems, as well as in computer games to imitate intelligent behavior [7, 17]. This algorithm creates a path from the available and unexplored nodes each time by selecting the vertex x with the lowest value of the cost function f(x), including the cost of going from the starting point to a specific point x and the cost of going from a specific point x to the destination point (estimated by heuristics). At each step, the algorithm appends the vertex with the lowest value of the function f(x) to the path and ends its operation when it encounters the vertex that is the destination vertex.

In computer games, methods based on artificial intelligence algorithms are also used to control the movement of vehicles, such as e.g. artificial neural networks. One of the first methods of artificial intelligence was the method related to the follow-up movement of the vehicle following the designated trajectory of its movement. Due to its limitations resulting

from the computation time, it was replaced with a newer method, which is currently widely used and is based on the system of checkpoints [1, 6]. Other currently used basic control methods include, among others: the Racing Lines method and the aforementioned control point system method.

The Racing Lines method was used in one of the first vehicle racing games, which was only abandoned when the computing power of computers increased so much that it could allow game designers to apply other more elaborate and time-consuming methods. This method also uses the trajectory of the car's movement on the racetrack, which the car has to follow. Line functions are used to draw trajectories, to which additional information is attached, such as the speed of the car [1, 6]. The advantage of this method is the low computational cost, the disadvantage is the limited ability to solve specific car journeys, and in some cases also providing an unrealistic solution.

The checkpoint method is the most common method where checkpoints are established along the racetrack. Cars then use the shortest path search algorithms to find the way to the next checkpoint. For a smooth avoidance of sharp bends, not the closest point but a few points ahead is used. Thanks to this solution, the cars look for the path to the point after the bend and have a sufficient distance to choose the appropriate trajectory as well as adjust their speed to changes [6, 15]. The disadvantage of this method is the need to manually mark checkpoints around the racetrack, which can be done in a semi-automatic way, i.e. the game developers themselves control the vehicles and thus set checkpoints on the racetrack map with their automatic registration in the appropriate database.

The Dynamic Pathfinding Algorithm (DPA) is an algorithm used in racing games to avoid obstacles on the racetrack. This algorithm uses two designated collision points that appear directly in front of the car at any given moment. Thanks to them, it is possible to detect obstacles on the right and left in front of the car. If both points simultaneously detect the barrier (the obstacle is right in front of the car), the collision points are set at an angle of 45o, i.e. in such a way that it is possible to find an empty space to turn the car in the right direction, avoiding a collision. This algorithm is used to solve complex situations that cannot be solved by other algorithms, hence it is used to avoid dynamic obstacles on the path found with another algorithm.

3. Possibilities of artificial neural networks in vehicle control

In recent years, as it was tried to show in the second section, various methods have been used to control vehicles in video games, to which artificial intelligence methods have recently

been added, including expert systems (e.g. decision trees), multi-agent methods and artificial networks. neural.

Decision trees used to control vehicle traffic are based on predetermined rules by which the vehicle is to follow. Such solutions are too rigid and therefore they are often the reason for the wrong course of the game. In this regard, high hopes are raised by artificial neural networks that learn to control the movement of vehicles in a flexible manner, hence an attempt was made to teach the Perceptron ANN to control vehicle traffic.

It is worth adding that artificial intelligence methods, such as expert systems, are often supplemented with procedural methods [14, 33], especially in terms of generating game-related content, and they are used to create terrain in various types of games, rather than the game itself. video games . Procedural methods certainly shorten the time of the algorithm, because they only require the preparation of appropriate components of the designed virtual world, but they are rigid solutions from the point of view of the player's capabilities on the track.

For the above-mentioned reasons, attempts were also made to create new games using the concept of multi-agent methods. In these methods, special objects (agents) have the ability to analyze the surrounding virtual terrain, hence they are able to modify it appropriately in accordance with the adopted rules, and thus become more flexible in adapting to the encountered difficulties. Therefore, in order to increase the flexibility of vehicle control on the racetrack, an attempt was made to use artificial neural networks in vehicle motion control. Artificial neural networks, despite their great potential to increase the flexibility of the course of the game, are still relatively unpopular methods used in video games.

One of the successful examples of their use is the 2010 game called "Supreme Commander 2". In this game, enemy troops are controlled by the Artificial Neural Network learned from the model of the object movement control system, therefore, when enemy troops meet, the Artificial Neural Network decides to take an appropriate action depending, for example, on the number of both parties [6, 19]. Another example of the use of the Artificial Neural Network is the game from 2001 called "Black & White" and its continuation from 2005 called "Black & White 2". The creators of this series of games, that is, Lionhead Studios, hired a scientist, Richard Evans, dealing with artificial intelligence methods, who received many prestigious awards for his design and its implementation in the form of a neural model for this game [6].

Therefore, as it results from the literature review, it is possible to teach the Artificial Neural Network a model of neural control of vehicle traffic in video games along any route. The advantage of this approach is that there is no need to set checkpoints on each map available in the game, as it was designed in the game from 2000 under the name Colin McRae Rally 2.0. In

this game, the computer opponents are steered by an Artificial Neural Network that has been taught with the use of input parameters such as road curve, distance from the turn, ground type, vehicle speed and parameters, etc. and the corresponding output values used in the steering [35].

3.1. Data measurement method for ANN training

An experiment enabling data measurement for learning and testing of an Artificial Neural Network was designed on the racetrack of a target racing game using semi-automatic vehicle control, i.e. with the possibility of manual measurements of parameters during a designer-controlled vehicle on a racetrack and automatically placing them in relevant database.

Then, after obtaining the appropriate input and output data in the experiment of creating a neural model of the vehicle motion control system in a racing game, it is assumed that the Artificial Neural Network can be learned, tested and validated. Thus, the training file and the testing file are the key to training the Artificial Neural Network model of the neural control system of a racing game.

The race track is assumed to be a closed non-linear track with uneven side edges. It was assumed that the vehicle has parameters such as length, width, distance between the axles, as well as the steering angle and the current speed of the vehicle. In addition, the vehicle is equipped with five distance sensors directed in the direction of the car's travel at different angles, while the distances read from the sensors and the current speed of the vehicle are important for the Artificial Neural Network.

The vehicle is represented by a game engine object that provides basic translation mechanisms independent of the current rotation in the coordinate system, and while driving, it uses calculations in two-dimensional space, taking into account such constants as: friction and grip. A very important constant is adhesion, which takes a smaller value above a certain limit speed, so that the car behaves realistically enough for the experiment. It should be added that the wheels are not treated as separate objects and the time required for their turning is not taken into account.

The trajectory used to obtain data for training the Artificial Neural Network of the model of the neural system of vehicle motion control on the racetrack has been specially diversified to capture various atypical cases. The measurements made during the test drive were saved using a function specially implemented for this purpose, which recorded and saved the results of the test drive measurements controlled by the designer to a text file, i.e. the current distances and

speed as input data and signals from the player as expected output data for ANN. Then the values of input and output ANN values were normalized by scaling them to the interval <0; 1>.

3.2. Selection of ANN parameters for the needs of research experiments

While reviewing the available types of artificial neural networks, it was decided to use the Perceptron ANN as a network very well suited to the approximation of functions. It is a multi-layer, unidirectional, artificial neural network, without feedback. When designing its architecture, it was established that it will have 14 input neurons due to the number of inputs to the ANN (13 quantities relating to the measurement of the distance between the vehicle and obstacles and the current speed of the vehicle) and two output neurons due to the number of measured parameters at the ANN output, is the steering and acceleration ratios.

Moreover, the presence of one hidden layer with a different number of neurons was established, as well as the corresponding functions of neuron activation, i.e. the sigmoidal function on the hidden layer (tansig) and the linear function on the output layer (purelin). The method of back propagation of errors was selected to teach the Artificial Neural Network of the neural model of the vehicle motion control system on the racetrack. In addition, the following tools were used to prepare the experimental environment, i.e. Godot Engine game engine and MATLAB and Simulink programming environment [8, 10, 25].

Due to the fact that for the professional implementation of the experiment, the selection of appropriate tools and technologies enabling the implementation of the developed experiment design is of key importance, the need to carefully meet such basic assumptions as, among others: creating an environment for the course of the experiment, designing and implementing vehicles to be placed in experimental environment, developing the ability to move vehicles in the experimental environment with appropriate interaction, developing the visual layer of the experiment, skillfully modeling and simulating the movement of vehicles using the laws of physics, and conducting test experiments to check the correctness of the game.

Designers and producers of video games believe that the commonly used technology that meets the above-mentioned assumptions and can be used in game engines is the framework technology with implemented basic mechanisms that repeat in each game together with an editor that enables the use of basic engine functions, such as e.g. 2D or 3D graphics rendering engine, physics engine providing collision detection, sound and scripts, etc. [5, 15, 18]. These types of engines available on the gaming market enable the preparation of a game project and its export to various platforms. Some of the most popular engines include: Unreal Engine, Unity, CryEngine, GameMaker, Godot, and many others [5, 8, 10-11, 18].

It is generally assumed that the choice of an engine should be guided by its implementation possibilities adapted to the intended game environment, but also by their orientation towards specific applications in racing games. An improperly selected engine can make the work of a designer very difficult, the effects of which will not be satisfactory, as exemplified by the Bioware studio used during the development of Mass Effect: Andromeda. Their chosen Frostbite engine was designed for a different type of game. The engine developed by DICE was compared to the F1 car, offering enormous power, but it was very difficult to get out of it and then use it in an appropriate way [6].

In the case of the experiment performed as part of this study, the engine named Godot Engine was selected due to its large implementation possibilities and relatively high convenience in using its functionality (Fig. 11). The Godot Engine is a powerful, cross-platform software for creating 2D and 3D games from a unified interface. Thanks to a comprehensive set of tools, users can focus on creating games without having to re-create the same functions [8, 10]. Fig. 2 shows the Godot engine editor, which uses OpenGL ES 3.0 and 2.0 to handle graphics, depending on the user's preferences. Version 3.0 allows better graphics, while version 2.0 has lower requirements and is suitable for games launched directly in web browsers. The editor is available for Windows, Linux and macOS platforms, and games can be exported to Windows, Linux, macOS, Android, iOS and web browsers thanks to WebAssembly.

Figure 2. Godot engine editor. Source: Steam website [8].

Projects in the Godot engine consist of scenes that have a hierarchical structure of nodes, with the primary node from which all other inherit properties is Node. Each scene has one root node, with the main types of nodes being a three-dimensional node (Spatial), a two-dimensional node (Node2D) and a Control Node responsible for the engine user interface. The control nodes were built, among others, engine editor. An important convenience is the ability to write scripts

in programming languages from the C language group (C / C ++, C #) or in a dedicated GDScript language (Fig. 3).

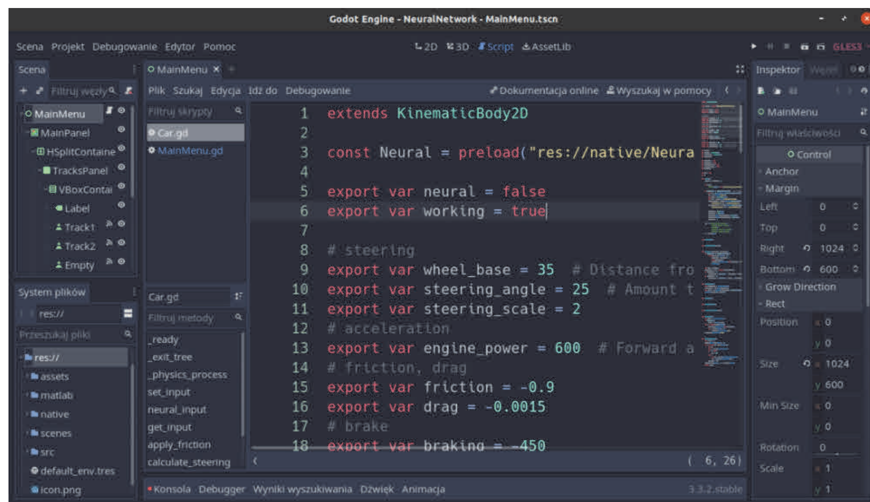

Figure 3. GDScript script editor. Selected markings: in the middle the editor window with a list of scripts and ´ functions on the left side. Source: [6, 8].

A supplement is a very accessible in designing editor VisualScript, which enables graphical designing of application behavior diagrams. As part of the created experiment, the GDScript language was used due to its strong orientation towards creating this type of application. In case of insufficient performance, nothing prevents you from combining it with a library written in C / C ++. In addition, GDScript is an object-oriented, dynamically typed language with a syntax similar to Python. Its goal is optimization and tight integration with the Godot engine. In addition, the engine editor allows you to import 3D scenes from many formats, including: glTF 2.0 (recommended), ESCN, FBX (experimental), Collada (.dae) and Wavefront OBJ (only static scenes). Thanks to this, you can use models prepared in external popular tools for modeling three-dimensional graphics. The experiment also used the environment of MATLAB and Simulink from Deep Learning Toolbox (the successor of the Neural Network Toolbox) to teach the Artificial Neural Network of a neural model of a vehicle motion control system on a racing track [6, 25]. The obtained program code in Matlab was generated using MATLAB Coder in C / C ++ in the form of a library that could be attached to the game engine.

4. Design and implementation of a racing game for vehicle traffic on a racetrack

4.1. Experimental conditions

It was assumed in the experiment that the designed racing game of vehicles has two race tracks shown in Fig. 4 and Fig. 5. It was also assumed that the vehicles (cars) driving on the

tracks shown in Fig. 6 have distance sensors directed to the front and sideways under different angles. After detecting a collision with another object, the sensor returns the distance in pixels. The movement of the car takes into account friction, air resistance, vehicle length and skid at speeds above the set limiting speed.

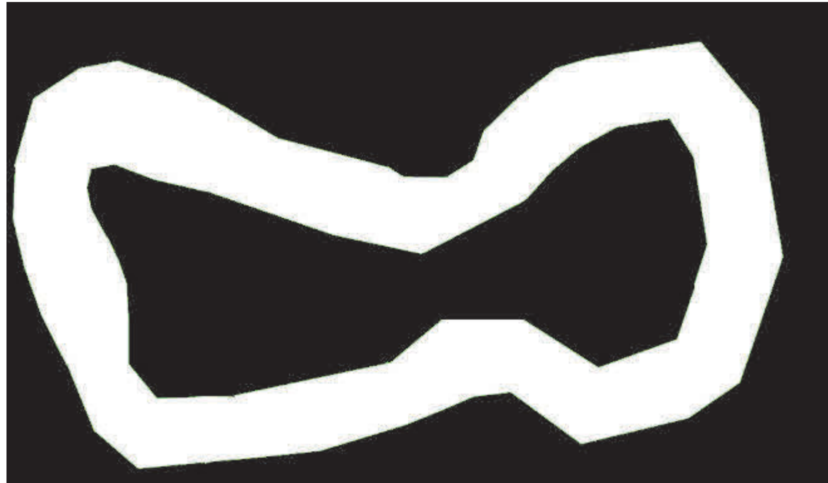

Figure 4. The first race track used in the experiment (so-called straight track). Source: [6].

Figure 5. The second race track used in the experiment (the so-called complex track). Source: [6].

4.2. Perceptron ANN as a neural model of a vehicle control system

The application of the vehicle motion on the racetrack includes: game design, virtual car design and the learned model of the vehicle motion control Perceptron SSN. The Artificial Neural Network was designed by Matlab using the Deep Learning Toolbox of the MATLAB and Simulink environment [6, 25, 29]. To design the ANN Perceptron architecture, 14 input quantities, two output quantities and one hidden layer with a variable number of neurons were used. The Artificial Neural Network script is shown in Listing 1.

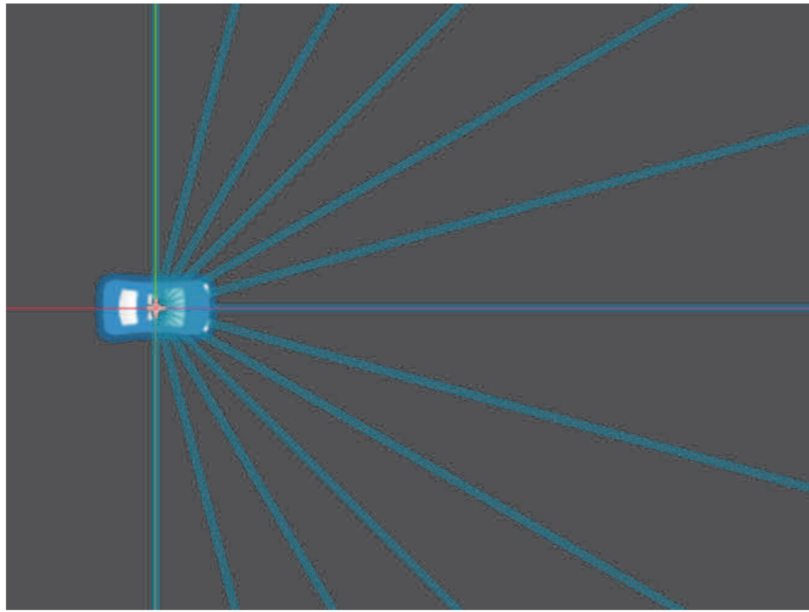

Figure 6. Car model used in the experiment. Source: [6].

```
% This script assumes these variables are defined:
%
% x - input data.
% y - target data.
trainFcn = 'trainlm'; %Levenberg - Marquardt Backpropagation.
% tor 2
x = x2';
t = y2';
hiddenLayerSize = 20;
net = fitnet (hiddenLayerSize, trainFcn);
```

Listing 1. MATLAB environment script generating and training SSN. Basic designations: x - matrix consisting of 14 input quantities to ANN (distances returned by 13 distance sensors and speed), a fragment of which is shown in Listing 2, y - matrix consisting of two output quantities from ANN (acceleration factor, turn factor) a fragment of which is shown in Listing 3. Source: [6, 25].

To train the Perceptron ANN, a database obtained by the designer in a semi-automatic manner (with registration of measurement points) was used, which consists of two files: a file containing input data and a file containing output data from ANN. Fragments of the standardized data file for ANN training and testing have been included in Listing 2 (ANN input data) and in Listing 3 (ANN output data), and the complete database has been presented in [6].

```
0.115668;0.120659;0.136791;0.168956;0.239415;0.372338;0.465447;...
0.115672;0.120663;0.136796;0.168961;0.239399;0.372261;0.465321;...
0.11568;0.120671;0.136805;0.16897;0.239371;0.372124;0.465095;...
0.115695;0.120687;0.136823;0.168988;0.239312;0.371837;0.464622;...
0.115719;0.120712;0.136851;0.169015;0.239223;0.371407;0.463915;...
```

```
0.11575;0.120744;0.136888;0.16905;0.239107;0.370842;0.462984;...
0.115788;0.120784;0.136932;0.169094;0.238963;0.370147;0.461838;...
0.115832;0.12083;0.136985;0.169146;0.238795;0.369327;0.46049;...
0.115883;0.120884;0.137046;0.169205;0.238602;0.368391;0.458947;...
0.115941;0.120943;0.137113;0.169271;0.238386;0.367342;0.45722;... ...
```

Listing 2. A fragment of the file with the input data. Source: [6].

```
0;0.490196
0;0.882353
0;1
0;1
0;1
...
```

Listing 3. The fragment of the file with the expected output. Source: [6].

Supervised learning was used to teach the Perceptron ANN because it works best in approximation tasks and is oriented towards teaching artificial multilayer networks, in which the architecture and parameters can be determined before starting the learning [9, 14, 16, 21, 23, 28-29, 38]. The development algorithms that modify the ANN structure during training, adapting it to the task conditions and data of the training file, as in the case of ANN changing its structure, are also promising. The method of back propagation of the neuron error begins with the calculation of the error δj on the j-th neuron in the output layer according to the relationship:

$$\delta_j = z_j - y_j , \tag{3}$$

where:

z_j - expected result from the training data on the j-th neuron of the output layer,

y_j – j-th exit from ANN.

Then the error propagates towards the entrance to the ANN using the relationship:

$$\delta_i = j\sum_{j=1}^{J} w_{ij}\delta_j , \tag{4}$$

where:

δ_i – error of the i-th neuron located in the current layer of neurons,

w_{ij} - the weight between the neuron and the current layer of neurons and the neuron j of the next layer of neurons,

δ_j - error of the j-th neuron from the next layer.

In the last step of the algorithm, the weights of each neuron are updated, determined on the basis of the calculated error δ according to the relationship:

$$\frac{dw_{ki}}{dt} = w_{ki} + \eta\delta_i \frac{df(net_i)}{dt} x_k, \tag{5}$$

where:

w_{ki} – new scale size,

η - learning coefficient,

δ_i – error on the i-th neuron,

$f(net_i)$ – activation function of the i neuron,

net_i - weighted sum of inputs to the i-th neuron,

x_k – k-th entrance.

Then, the learned and tested Perceptron ANN was converted and compiled into a static library in C ++ using the MATLAB Coder module [6, 25]. The Perceptron ANN generated in this way was added to the dynamic library, which contains the code constituting the link between the neural network and the functions provided by the game engine, and thus was included in the experiment design.

```
function[y1] = gdNet3T40N(x1)
% GDNET3T40N neural network simulation function .
% Auto - generated by MATLAB, 18 - Aug -2021 19:46:31.
% [y1] = gdNet3T40N (x1) takes these arguments :
% x = 14 x Q matrix, input #1
% and returns :
% y = 2 x Q matrix, output #1
% where Q is the number of samples.
%# ok <* RPMT0 >
% ===== NEURAL NETWORK CONSTANTS =====
% Input 1
x1_step1.xoffset = [0.014469;0.014929;...];

% Layer 1
b1 = [22.151098785002407254;9.5825260403490748473;...];
IW1_1 = [ -1.1876712196389531684 9.2544919281221016405 ...];

% Layer 2
b2 = [2.87203967630265522;1.0374897587669904553];
LW2_1 = [0.18254985882101260053 -0.33403912180241673857 ...];
```

```
% Output 1
y1_step1.ymin = -1;
y1_step1.gain = [1;2];
y1_step1.xoffset = [ -1;0];

% ===== SIMULATION ========
% Dimensions
Q = size ( x1 ,2); % samples
% Input 1
xp1 = mapminmax_apply( x1 , x1_step1 );

% Layer 1
a1 = tansig_apply(repmat( b1 ,1 , Q ) + IW1_1 * xp1 );

% Layer 2
a2 = repmat ( b2 ,1 , Q ) + LW2_1 * a1 ;

% Output 1
y1 = mapminmax_reverse ( a2 , y1_step1 );
end

% ===== MODULE FUNCTIONS ========
% Map Minimum and Maximum Input Processing Function

function y = mapminmax_apply (x , settings )
y = bsxfun ( @minus ,x , settings . xoffset );
y = bsxfun ( @times ,y , settings . gain );
y = bsxfun ( @plus ,y , settings . ymin );
end
% Sigmoid Symmetric Transfer Function
function a = tansig_apply (n ,~)
a = 2./(1 + exp( -2* n )) - 1;
end
% Map Minimum and Maximum Output Reverse - Processing Function
function x = mapminmax_reverse (y, settings )
x = bsxfun (@minus ,y , settings.ymin );
x = bsxfun (@rdivide ,x , settings.gain );
x = bsxfun (@plus, x, settings.xoffset );
end
```

Listing 4. A function serving as a model of a neural vehicle motion control system in a racing game, generated with the use of MATLAB Coder. Source: [6].

4.3. Virtual car

The Perceptron ANN was used as a neural model of the car motion control system (Fig. 6), which was finally equipped with 14 inputs, including 13 distance sensors and one quantity representing the current speed of the car, with one of the sensors pointing perpendicularly from the center of the car's forehead. to the front of the car, and the other sensors to the sides symmetrically distant from it at another angle which is a multiple of 15°, i.e. 15°, 30°, 45°, 60°, 75° and 90°.

It was assumed that the car object extends the KinematicBody2D class providing appropriate methods for moving and rotating the object in two-dimensional space. This class is intended for player-controlled objects. When each image frame is rendered, the _physics_process (delta) method in Listing 5 is called, where delta is the time elapsed since the previous frame.

```
func _physics_process ( delta ):
  acceleration = Vector2 . ZERO
  if ( neural ):
  neural_input ()
  else :
  get_input ()
  apply_friction ()
  calculate_steering ( delta )
  velocity += acceleration * delta
velocity = move_and_slide ( velocity )
```

Listing 5. Physics_process (delta) method called when calculating the physics of an object. Source: [6].

When putting a car on the track, a flag is set by which the _physics_process (delta) method gets the control parameters from the player or from the ANN. Control parameters are quantities in the range <−1; 1> acting as a coefficient multiplied by the maximum steering angle and braking speed or force, which are scaled constants.

Then these parameters are passed to the set_input (turn_param, accel_param) (Listing 6) method, which creates the acceleration vector on their basis and can save the measurement data.

```
func set_input ( turn_param , accel_param ):

  if Global . mode == Global . MEASURE_MODE :

 if abs ( velocity.length ()) > 0:
  file_count = file_count + 1
```

```
    print ( file_count )
    result_file . store_string ( str ( turn_param ) + ";"
    + str ( accel_param )+"\ n ")
    write_data ()
    var turn = turn_param
    steer_angle = turn * deg2rad ( steering_angle )

  if accel_param > 0:
    acceleration = transform . x * engine_power * accel_param

  if accel_param < 0:

 acceleration = transform . x * braking * ( - accel_param )
```

Listing 6. Set_input (turn_param, accel_param) method to get control parameters. Source: [6].

The apply_friction () method (listing 7) modifies the current velocity vector of the drag and friction coefficients.

```
   func apply_friction ():

    if velocity.length () < 5:
    velocity = Vector2.ZERO
    var friction_force = velocity * friction
    var drag_force = velocity * velocity.length () * drag

   if velocity. length () < 100:
    friction_force *= 3
    acceleration += drag_force + friction_force
```

Listing 7. The apply_friction () method for calculating friction and drag forces. Source: [6].

The last method calculate_steering (delta) (listing 8) is responsible for rotating the velocity vector and the car object itself. Finally, the car is moved along the velocity vector using the built-in move_and_slide (velocity) method.

```
 func calculate_steering (delta):
    var rear_wheel = position - transform . x * wheel_base / 2.0
    var front_wheel = position + transform . x * wheel_base / 2.0
    rear_wheel += velocity * delta
    front_wheel += velocity . rotated ( steer_angle ) * delta
    var new_heading = ( front_wheel - rear_wheel ). normalized ()
    var traction = traction_slow

    if velocity . length () > slip_speed :
```

```
traction = traction_fast
var d = new_heading . dot ( velocity . normalized ())

if d > 0:
velocity = velocity . linear_interpolate ( new_heading * velocity . leif d < 0:
velocity = - new_heading * min ( velocity . length () , max_speed_reverrotation =
new_heading.angle ()
```

Listing 8. Apply_friction() method to calculate the rotation of the car. Source: [6].

5. Conclusions

This paper shows that the Perceptron Artificial Neural Network can be used as a neural model for controlling vehicle movement along the racetrack. For the purpose of obtaining a training and testing file, a research experiment was designed to enable measurements of the appropriate number of points on the racetrack. The learning pairs obtained were used in teaching and testing the Artificial Neural Network of the neural model of the car control system along a given race track.

In order to meet the requirements resulting from the assumed conditions of research experiments, it was necessary to design an appropriate ANN architecture and select appropriate parameters for it, and above all the number of hidden layers. It turned out that one hidden layer with the number of neurons from 20 to 40 is sufficient, depending on the conducted research experiment. The tansig function for the hidden layer and the purelin function for the output layer of neurons were used as the function of neuron activation.

In experimental studies, the key issues turned out to be, among others: the number of training pairs, i.e. measuring points on the racetrack, the way of initiating neuron weights, the results of the analysis of testing and simulations checking the movement on the racetrack with the use of, as well as the interpretation of the obtained results and their translation to improve the parameters of the experiment and the artificial neural networks used in it.

During the construction of the ANN, 14 input quantities and two output quantities were used. The inputs were the distances returned by 13 sensors placed on the car and the current speed of the car, and the outputs were the acceleration coefficient and the steering coefficient of the vehicle. The programming environments consisted of such components as: CLion environment, Godot game engine and MATLAB computing environment [8, 10-11, 25]. The aforementioned tools made it possible to efficiently perform, that is, to design and implement research experiments, as well as to test them later and conduct simulation studies.

As could be expected, the first experiments were burdened with certain failures, consisting in the collisions of the car with the walls of the racetrack. As a result of testing the learned ANN model of the neural vehicle motion control system, that the main cause of the collision and the lack of complete influence on the car control was too small number of training and testing pairs. It turned out that after introducing a vastly large number of training pairs, the designed and implemented ANN met the requirements for flexible control of vehicle movement on the racetrack.

As a result of simulation tests, it turned out that the longest lap of the track in the conducted experiments lasted 4 minutes and 55 seconds, and the shortest - 10.47 seconds. In five minutes, the highest number of laps was 34, while the lowest numbers of laps were 1 and 5. In the course of the experiments, shortcomings of the neural network were noticed, which consisted in the fact that under the same conditions the learning outcomes may be different.

You can further improve both the ANN learning method itself, and you can further increase the accuracy of learning with other artificial intelligence methods, such as evolutionary algorithms [4, 14, 30], or instead of the Artificial Neural Network to control the movement of the vehicle, you can use e.g. swarming algorithms such as such as the ant algorithm [22, 26, 29, 34], firefly algorithms [27], or the systemic immunological algorithm for increasing the ANN resistance to wall collisions [36]. Developing ANNs can also be used, i.e. ANNs that change their structure [13, 31-32].

References

1. Abdal M.: Artificial Intelligence in Racing Games. University of Birmingham. https://www.cs.bham.ac.uk/~ddp/AIP/RacingGames.pdf [access: 2021-04-16].

2. Algorytm BFS. https://eduinf.waw.pl/inf/alg/001_search/0126.php [access: 2021-04-16].

3. Algorytm Dijkstry. http://www.algorytm.org/algorytmy-grafowe/algorytm-dijkstry. html [access: 2021-04-16].

4. Arabas J.: Wykłady z algorytmów ewolucyjnych (Eng. Lectures on evolutionary algorithms). WNT. Warszawa 2003, pages 303.

5. Barczak A. and Woźniak H.: Comparative Study on Game Engines. Studia Informatica. Systems and Information Technology 1-2(23)2019.

6. Bolesta A.: Artificial Neural Networks as vehicle control systems in racing games. Master's thesis written under the supervision of dr hab. inż. Jerzy Tchórzewski, prof. uczelni w Instytucie Informatyki, na kierunku informatyka na Wydziale Nauk Ścisłych i Przyrodniczych, UPH w Siedlcach, Siedlce 2021, pages 82.

7. Cui X. and Shi H.: Direction Oriented Pathfinding In Video Games, International Journal of Artificial Intelligence & Applications, 2 Oct. 2011.

8. Dokumentacja silnika Godot (Eng. Documentation of the Godot engine), https://docs.godotengine.org/pl/latest/ [access: 2021-04-16].

9. Flasiński M.: Wstęp do sztucznej inteligencji (Eng. Introduction to Artificial Intelligence). WN PWN, Warszawa 2011.

10. Godot na platformie Steam. URL: https://store.steampowered.com/app/404790/Godot%5C_Engine/ [access: 2021-04-16].

11. Graham R., McCabe H., and Sheridan S.: Neural Networks for Real-time Pathfinding in Computer Games. Jan. 2004. Środowisko CLion. https:// www.jetbrains.com /clion/?gclid=CjwKCAjw8cCGBhB6EiwAgOReyyEyTBD5gq4mvxeo4vMWqwDXjk0wpBxnIIVewVGuPop667dqTgDLRoC3P4QAvD_BwE&gclsrc = aw. ds. [access: 2021-04-16].

12. Horzyk A.: Metody Inżynierii Wiedzy – Uczenie głębokie i głębokie sieci neuronowe (Eng. Knowledge Engineering Methods - Deep learning and deep neural networks), AGH w Krakowie, Warszawa 2019, http://home.agh.edu.pl/~horzyk/lectures/miw/MIW-DL.pdf [access: 2021-04-16].

13. Kłopotek M., Tchórzewski J.: The concept of discoveries in evolving neural net, Advances in Soft Computing, IPI PAN, No. 17, Warszawa 2002, pp. 165-174.

14. Mulawka J.: Systemy ekspertowe (Eng. Expert systems), WNT, Warszawa 1996, pages 235.

15. Oliveira M. M., Chan M. T., Chan C. W., and Gelowitz C.: Development of a Car Racing Simulator Game Using Artificial Intelligence Techniques, International Journal of Computer Games Technology (Nov.), p. 839721, 2015 [access: 2021-04-16].

16. Osowski S.: Sieci neuronowe do przetwarzania informacji (Eng. Neural networks for information processing). OW PW, Warszawa, pages 422, 2013.

17. Patel A.: Introduction to A*, URL:http://theory.stanford.edu/~amitp/Game Programming/AStarComparison.html [access: 2021-04-16].

18. Przychodzki M: Neuro-evolution of artificial neural networks based on the NEAT algorithm. Master's thesis written under the supervision of dr Artur Niewiadomski, kierunek informatyka, Wydział Nauk Ścisłych i Przyrodniczych, Uniwersytet Przyrodniczo-Humanistyczny w Siedlcach, Siedlce 2021.

19. Robbins M.: Neural Networks in Supreme Commander 2, https://ubm- twvideo01s3.amazonaws.com/o1/vault/gdc2012/slides/Summit_AI/Robbins_Michael_Off%20the%20Beaten.pdf. [access: 2021-04-16].

20. Ruciński D.: The Influence of the Artificial Neural Network type on the quality of learning on the Day-Ahead Market model at Polish Electricity Exchange join-stock company. Studia Informatica. Systems and Information Technology. Vol. 1-2(23)2019, pp.77-94.

21. Rutkowski L.: Metody i techniki sztucznej inteligencji (Eng. Methods and techniques of artificial intelligence). WN PWN, Warszawa 2017.

22. Sitkiewicz T., Tchórzewski J.: Wykorzystanie algorytmów mrówkowych do poprawy funkcjonowania algorytmu ewolucyjnego dla zagadnień transportowych (Eng. The use of ant algorithms to improve the functioning of the evolutionary algorithm for transport issues), Zeszyty Naukowe AMW, Nr 169 K/1, pp. 349-362, 2007.

23. Tadeusiewicz R. and Szaleniec M.: Leksykon sieci neuronowych (Eng. Lexicon on Neural Networks), Wydawca Projekt Nauka, pages 134, Jan. 2015.

24. Strona główna silnika Godot (Eng. The home of the Godot engine), https://godotengine.org/ [access: 2021-04-16].

25. Strona główna MATLAB & Simulink, https://www.mathworks.com/products/matlab.html [access: 2021-04-16].

26. Szewczak M., and Trojanowski K.: Wirtualne laboratoria optymalizacji heurystycznej: wykorzystanie algorytmów mrówkowych (Eng. Virtual heuristic optimization laboratories: the use of ant algorithms). Studia Informatica. Systems and Information Technology 1-2(11)2003, pp. 87–100.

27. Świtalski P., Bolesta A.: Firefly algorithm applied to the job-shop scheduling problem. Studia Informatica. Systems and Information Technology. Vol. 1-2(25)2021, pp.87-100.

28. Tadeusiewicz R.: Elementarne wprowadzenie do techniki sieci neuronowych z przykładowymi programami (Eng. Elementary introduction to the technique of neural networks with sample programs). Problemy Współczesnej Nauki: Informatyka. AOW, 1998, https://books.google. pl /books?id=SjgzygAACAAJ [access: 2021-04-16].
29. Tchórzewski J.: Metody sztucznej inteligencji i informatyki kwantowej w ujęciu teorii sterowania i systemów (Eng. Methods of artificial intelligence and quantum computing in terms of control theory and systems), Wydawnictwo UPH w Siedlcach, 2021, pages 343.
30. Tchórzewski J.: Systemowy Algorytm Ewolucyjny SAE (Eng. Systemic Evolutionary Algorithm), Bio-Algorithms and Med-Systems, Vol. 1, No. 1/2, 2005, pp. 61-64.
31. Tchórzewski J., Kłopotek M.: A Case Study in Neural Network Evolution, Prace Naukowe Instytutu Podstaw Informatyki PAN, Nr. 943, IPI PAN, Warszawa 2002.
32. Tchórzewski J., Kłopotek M.: The Concept of Making Discoveries in Evolving Neural Net, Intelligent Information Systems 2002, Physica-Verlag HD, pp. 165-174.
33. Tchórzewski J.: Systemy ekspertowe (Eng. Expert Systems), [w:] Użytkowanie mikrokomputerów IBM PC. Część II. Podstawowe oprogramowanie, [pod red. Tchórzewski J., Barczak A., Barański M., Rozwadowski L.], Wydawnictwo WSR-P w Siedlcach, Siedlce 1993, pp. 131-177.
34. Trojanowski K.: Metaheurystki. Materiały pomocnicze do przedmiotu “Metaheurystyki – laboratorium” (Eng. Metaheurists. Auxiliary materials for the subject "Metaheuristics - laboratory"). Wyd. WSISiZ, Warszawa 2003, pages 80.
35. Wardziński K.: Przegląd algorytmów sztucznej inteligencji stosowanych w grach komputerowych (Eng. Review of artificial intelligence algorithms used in computer games), Homo communicativus. Filozofia – komunikacja– język – kultura (3 May): Kulturotwórcza funkcja gier. Cywilizacja zabawy czy zabawy cywilizacji? Rola gier we współczesności, pp. 249–263, 2008.
36. Wierzchoń S.: Sztuczne systemy immunologiczne. Teoria i zastosowania (Eng. Artificial immune systems. Theory and Applications). AOW EXIT, Warszawa 2001, pages 282.
37. Yannakakis G. N. and Togelius J.: Artificial Intelligence and Games. Springer, 2018.
38. Żurada J., Barski M., and Jędruch W.: Sztuczne sieci neuronowe: podstawy teorii i zastosowania (Eng. Artificial neural networks: basic theory and applications). WN PWN, Warszawa 1996, pages 375.