Imitation of Human Behavior in 3D-Shooter Game

Makarov Ilya, Tokmakov Mikhail, Tokmakova Lada

National Research University Higher School of Economics,
Department of Data Analysis and Artificial Intelligence
iamakarov@hse.ru
matokmakov@gmail.com
lrtokmakova@yandex.ru

Abstract. There are many algorithms for 3D-shooter game artificial intelligence that are based on automatic target recognition with the help of ray casting, script-based event systems and many others. In such models game agents have an advantage over human player because computers do not need time for enemy recognition and target aiming processes. The main goal of our research is to imitate a realistic enemy player under PC control, which acts similar to a human reaction. We consider such aspects of shooter in-game processes as visual recognition of players, the delay for the response on each game event and distributed strategies of decision making. The paper presents method of visual recognition of player's 3D-model with the use of geometry knowledge of the three-dimensional maze and target sighting with respect to human motor reflexes. The implementation is made in Unreal Engine 4, which provides large variety of methods for programming game agents.

Keywords: Game Artificial Intelligence, Neural Network, Visual Recognition, Sparse Network of Winnows, Boosting, Rule-based Systems.

1 Introduction

Entertainment industry provides a vast variety of different games, every time attempting to improve realism of virtual world. It involves modeling of an enemy under automatic computer control (so called, BOT) having reaction that is similar to human reaction. In fact, the user has to spend some time for enemy visual recognition, and adapt motor reflexes for target sighting of proper level. These circumstances together with the use of ray-casting algorithms allow an enemy to obtain immediate perfect target aim without even small time delays and sighting errors.

Multiple algorithms are presented to compensate BOT supremacy over human player, such as simulated time delay to fire back, bad accuracy during shooting process, almost infinite health level and infinite regeneration of human player and many other methods to retain balance between entertainment and game challenge in virtual world. We focus on creating game artificial intelligence agent for a 3-D shooter game imitating human-like behavior.

The article represents a new approach to the problem of creating realistic game agent for 3D-shooter. We try to find a solution to the inverse problem using machine and

neural learning algorithms to obtain maximum likelihood estimators of target correction parameters and enemy detection. We also make statistical experiment to find corresponding clusters of such delays relative to skill-level of human player.

Implementation is made using modern technologies of game agent modeling, which allow us to combine different schemes and algorithms into one common behavior tree algorithm, saving the properties of complementarity and interchangeability of the individual modules for decision making models and different visual recognition patterns.

The Blueprints visual scripting system in Unreal Engine 4 is a complete gameplay scripting system based on the concept of using a node-based interface to create gameplay elements from within Unreal Editor.

With the use of Blueprints, programmers can implement and modify any gameplay element:

- Gameplay conditions and rules.
- Developing diverse meshes and *Materials* or character customization.
- New camera perspectives and change dynamically the camera view.
- Items - weapons, pickups, med kits, triggers, and others.
- Environments - create procedurally-generated items.

Blueprints combine the best properties of previous AI models and propose sufficient interface to create complex models in short time.

Finally, we may say that developers frequently use existing models of computer player's behavior, which sometimes do not allow to achieve sufficient realism. In different games we may have texture issues, bad organized respawn, wanton shooting and the worst – disadvantage of human player in comparison with game AI due to widespread usage of engine algorithms in decision making algorithms. Our work is an attempt to form our own pattern of AI, which will combine human and machine visual perception and decision making models to implement human-like BOT.

Real-time game play has always been major setback for AI in video games. If the action continues arbitrary long, there will be no time to compute all possible actions and future states. In order to achieve diverse BOT difficulty levels a specific model should be constructed, providing computational part for:

- Confidence intervals for time delay and sighting accuracy using grade of visual recognition;
- Target sighting on fire recognition with adjusted parameters;
- Statistical model for the curves of the first (initial) and the second (correction) targeting accuracy and delays;
- Adaptation of shooting process with respect to recoil;
- Rule-based decision making system for priority actions with respect to a map of game events.

The first part of the paper is brief review of existing games. We analyze the history of BOT developing and snow minuses of automatically visual recognition. The second part of the article considers methods of image processing with the help of geometry

knowledge of three-dimensional maze. We present a statistical prediction model to obtain possible dangerous directions where an enemy appearance may occur more frequently and we organize some data structure to process these frequencies as priority goals for decision making module. The third part of our article describes basic algorithms of visual recognition and their applications to our problem relative to practical success in terms of time complexity and percent of successful enemy recognition. We deal with the description of samples' properties, and use neural network for frequencies, obtained from the geometry of a maze, to increase the possibilities for enemy recognition in dangerous zones. It does improve the quality of recognition in difficult places such as corners, box's covers, and does not affect for enemy recognition in front of a player, where the recognition method gives good results by itself. The next part of the paper deals with the decision making model and represents a combination of two mentioned above approaches into one AI algorithm. The article ends with the description of gaming process.

2 Brief Review of Existing 3D-shooters

The first "3-D action" was created in 1992 with the use of «Wolfenstein 3D» engine. The game has the same name, and visualization of 3D map and models was achieved by ray casting. This method emits one ray for each column of pixels, evaluates to see whether it has intersection with a wall and draws textures creating a one-dimensional Z-buffer. The game has widely popularized the genre on the PC and has found the basic run-and-gun type for many subsequent first-person shooter games (FPS games).

The game mentioned above was not as popular as its analog "Doom", which was developed in 1993. It was based on «Doom engine», which belongs to the same series of engines like «Wolfenstein 3D». In comparison with Wolfenstein 3D the Doom engine includes such features as non-perpendicular walls, full texture mapping of all surfaces, dynamic platforms and floors. In addition to ray casting the BSP tree technology created by Bruce Naylor was used. The lifelike environment was achieved by using the stereo sound system, which made it possible to quickly determine the direction and distance of a sound effect made by an enemy or occasional environment event. The player has to be on guard while he hears monsters' roars, and receives additional knowledge to find new areas in the form of sounds of some door which open remotely. Enemies can also become aware of the player's presence by hearing distant gunshots. There were also popular features of the Doom engine, such as the ability to create custom scenarios contributed significantly to the game's popularity.

It was the first step of FPS gameplay evolution. Nowadays the market of computer games is overcrowed. An artificial intelligence of many shooters has several shortcomings because developers concentrate more efforts on visual effects and entertainment, whereas BOT action can be not realistic. We shall now move to brief review of several modern games relative to artificial intelligence models and their properties in the context of setting goals for BOT.

One of the most popular FPS games, "Call of duty: Ghosts", was published in 2013 [1]. Despite the high-quality graphic and large variety of cinematic effects CoD series

continued modern tradition to publish AAA projects without sufficient testing procedures. One's virtual players can shoot the ceiling trying to kill enemies, which are on the floor above. Also, some adversaries can rest against the wall and not notice User in two steps. There is also a famous bug migrating in the CoD series: after the player's death reloading last checkpoint activates all enemy AIs without even appearing of player in the viewing angle. Some bugs appeared from wrong scripts, but many others involve low-level command block and determined algorithms, which do not cover all gameplay opportunities. One of the things that were highlighted during the Call of duty: Ghosts presentation was the feature that fish AI moves out of the way when user gets close. Actually, this feature is not even an innovation — it was nearly 20 years old!

The so-called "NextGen" notion for next generation of games is apparently relevant only for the graphical part of the games, but the vision of AI models is in creating tools for different games and simple developer kit but not for peculiar game properties.

"Counter-Strike: Global offensive" was developed in 2012. BOTs' interaction with outside world is not realistic in some cases, for example, enemies can jump from the second floor and their level of health will not change. However, to support the many community-created maps, CS:Source and CS:Global Offensive include an automatic mesh generation system. The first time you play a custom map with BOTs, the generation system will build a .nav file for that map. Depending on the size and complexity of the map, this process may go for a few minutes to a few hours.

The following steps occur during Navigation Mesh generation:

- Starting at a player spawn point, space for walking is sampled by "flood-filling" outwards from that spot, searching for adjacent walkable points
- Right-Angled Navigation Areas are built from the sampled data
- Hiding and Sniper spots are computed
- Encounter Spots and Approach Points are calculated (this can take awhile)
- Initial Encounter areas are computed

There are general mesh problems, for example, extraneous areas on stairs and ramps, railings, preventing excessive jumping, dealing with rotating doors.

BOTs will sometimes try to walk through the opened door brush from one nav area to another and get stuck.

Although mesh navigation had leading positions for a few years there were many issues with moving scene, extreme position cases, so a new concept for programming BOT action has appeared.

Summing up, we see that automatically enemy detection can lead to different types of errors in shooter games. It does not represent realism, for instance, identification of enemies through the walls and appearance of BOT in textures. The research aims to create own detection module in order to avoid mentioned above mistakes.

3 Geometry

We created BOT in Unreal Engine 4. The order of its actions depends on the objects encountered on its way. The appearance of the virtual player is a template shown in the picture:

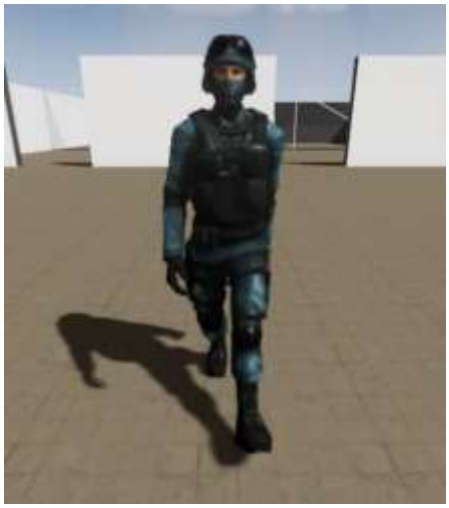

Fig. 1. Screenshot of BOT

We consider the one-story maze with the following objects:

- Walls
- Doorways
- Boxes
- Columns

The order of BOT's actions is a priority queue, which is denoted by S. When our virtual player faces the corners in the maze, S updates. All mentioned above objects have several corners. First of all, we calculate automatically mesh navigation to determine corners and separate possible fast scene changing. Then we process the queue considering a set of neighbors:

1. finding element with maximal priority
2. recalculating danger zones with respect to impact of neighbors by geometrical closeness with respect to mesh navigation
3. comparing the first K queue elements by using decision making module.

The constructed model allows us to separate geometrical combination of danger zones by Euclidian distance and different priorities in BOT reaction.

Let's look at an example illustrated in the picture:

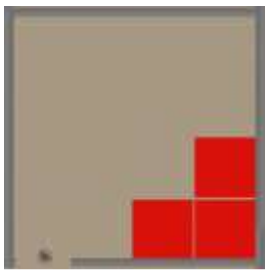

Fig. 2. Around the corner

We obtain the event “look around the corner” skipping visual recognition of the objects as if we already know them. We find out similar events with nearby distance from the current corner and include them in our calculations. The situation is separated by three simple steps. The first of them is the case in which BOT meets the nearest corner (the right side of the doorway). Then we process boxes as consequent corners in our model with respect to their priority and distance. The probability to turn on the angle X from 0 to 90 degrees with respect to the right wall is represented by density functions. A little circle represents BOT position.

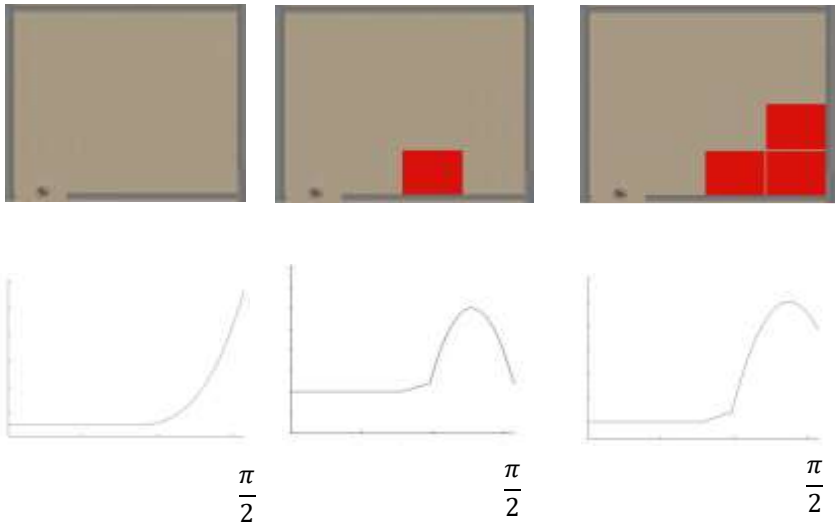

Fig. 3. Processing priority queue when looking around the corner

The obtained density functions represent the process of visual recognition of objects when we search for an enemy but have to spend some time on processing object’s shapes. Using geometry knowledge we reduce amount of time on visual recognition of objects using its statistical equivalent to find dangerous zones as equivalent of comparing visual images of objects and an enemy.

4 Visual Recognition

Visual recognition is the assignment of given pictures to a particular class of objects with the help of known methods of classification. We implement three standard steps of it:

- The converting of the source image to the initial idea;
- The extracting characteristic values of recognition;
- The classification of the image to a certain class with the use of the classifier.

In our work we consider three basic approaches to the visual recognition:

- *Heuristic methods* [2, 3]
    - *Full heuristic model*. Expert compiled a set of rules that describe the image of the object (built model), according to which detection is produced.
    - *Search for characteristic invariant attributes*. It is the description of characteristics of the object, which are invariant with respect to possible distortions (lighting changes, rotation, scaling).
- *Method of pattern matching*. It is the creation of a template for the image of the whole object or its characteristic features. The pattern can be a complicated structure and allows for different deformations and transformations. This approach allows implementation in real time.
- *Methods of learning by precedents.* The model is automatically built on the set of images of an object compiled beforehand from the possible input data of the system [4, 5, 6, 7].

The latter approach is the most interesting for us, because results of testing process show that such methods are more appropriate, so let's consider the last approach in detail. It consists of two units: converting the image into a feature vector and classification. The former means the most complete and informative representation of an image in the form of a numerical vector. The latter fact implies the verification of hypothesis accessories image to the class object images based on the observation that is the feature vector.

Denote a feature vector for $x \in X \subseteq R^n$ - the description of the object, which is output of the conversion unit. Class will be called a subset $K_y = \{x \in X, y^*(x) = y\}$ of the set X, $y \in Y \subseteq Z$ – the set of the class markers. We will view the case of binary classification (the image is an image of the desired object or the image is an image of anything else), consequently $Y = \{-1;\ 1\}$. $y^*(x)$ – the mapping, which is defined for all $x \in X$ and is specified the partition into subsets K_y. It is worth noting that classes K_y can overlap, because the feature vector is only a description of an object's attributes, and two different images can have the same specifications. The training set is pairs of precedents $T = \{(x_1, y_1), \dots, (x_l, y_l)\}: y^*(x_i) = y_i. i = \overline{1, l}.$

For the application of classification algorithms and pattern recognition we make the following hypothesis: the set $X \times Y$ is a probability space with probability measure P,

precedents $(x_1, y_1), \dots, (x_l, y_l)$ appear randomly and independently in accordance with the distribution P.

The task of classification is to build the function $F(x)$, which approximates the mapping $y^*(x)$ with the help of the training set.

Let's consider several methods of learning by precedents, which we tested in our work.

4.1 Bayesian Classification Methods

The first method is the principle of posteriori probability maximum [8]. It is based on following hypothesis:

- $X \times Y$ is the probability space with probability measure P, precedents $(x_1, y_1), \dots, (x_l, y_l)$ appear randomly and independently in accordance with the distribution P.
- We know density distributions of classes $p_y(x) = p(x|K_y), y \in Y$, which we will call likelihood functions.
- We know the probability of occurrence of each object from classes $P_y = P(K_y), y \in Y$, which we will call the priori probabilities.

Decision rule can be written in the following form:

$$F(x) = arg \max_{y \in Y} P(K_y|x) = arg \max_{y \in Y} p_y(x) P_y \quad (1)$$

The second method is called Bayesian networks [9]. This approach is based on the combination of the principle of posteriori probability maximum with graph theory. The idea is to create a graph with vertices corresponding to any component of the feature vector, and edges indicating a causal relationship.

4.2 Classical Neural Networks

The main idea is the serial conversion of a signal with the elementary functional elements (neurons), operating in parallel [10]. The basic principle of neural network configuration is to use optimization methods to minimize the mean square error. Also, the neuron networks are capable of retraining.

4.3 SVM - Support Vector Machine

The SVM algorithm constructs a linear separating surface (hyperplane), equidistant from the convex hulls of the classes, convex hull is based on the precedents [11]. If the hyperplane does not exist (classes are not linearly separable), sound conversion will apply for nonlinear classification, which projects X in the space of higher dimension, probably, infinite. The algorithm for constructing the classifier is reduced to the problem of quadratic programming. The solution of this task will be unique and the found extremum will be global.

4.4 SNoW - Sparse Network of Winnows

Sparse network of Winnows is a special type of neural networks [12]. The network consists of two (the number of possible classes) of linear neurons associated with the feature vector components. Geometrically, SNoW consists in two hyperplanes in the space of feature vectors. Vector belongs to the class, corresponding to which the hyperplane is the closest. The resulting separating surface is thus a hyperplane in the original space X.

4.5 Classifier Boosting

This approach combines primitive classifications into one stronger method. The main idea is an iterative minimization of a convex functional classification error. Boosting attempts to project the original data into a space where the classes are linearly separable.

4.6 Results

In our research we obtained the following results, using data from demorecords of multiplayer game sessions:

- SNoW and boosting distinguish the highest percentage of correct detections, about 94% [13, 14];
- SVM, SNoW and boosting provide high speed of recognition;
- SNoW and boosting have low level of the second kind error.

The correctness of visual recognition was manually checked. Thus, at the current level of experiment the most appropriate method for our problem is the combination of two methods: Sparse network of Winnows and Classifier boosting.

The picture below represents the result of visual recognition unite, using classifier boosting.

Fig. 4. Result of visual recognition

In this situation there are two objects, which can be recognized as enemies: the virtual player and its shadow. But the geometry unit defines that the latter is on the floor, so AI focuses only on the first one.

5 Decision Making Model

In the most basic games, there was simply a lookup table for what was currently happening and the appropriate best action. In the most complex cases, the computer would perform a short minimax search of the possible state space and return the best action [15]. The minimax search had to be of short depth since the game was occurring in real-time.

We use the blueprints visual scripting system in Unreal Engine 4 [16, 17]. The node-based interface allows us to create the maze with different objects (for instance, boxes, doors). Also, this concept gives an opportunity to make any appearance of BOT and to determine the direction of the eye with the help of a camera-element. The virtual player is the combination of several blueprints, each of which corresponds to the certain action: moving, shooting, behavior during shooting and search for enemies.

We focus on a rule-based system (RBS) [18, p.169] to identify simple situations and make a choice for a complex processes as a tuple of simple rules. RBS works with instructions and the interpreter. In our case, rules are oriented tree structures. Let's discuss the short list of game events:

1) Walking
    a) Bypassing and Search
    b) Moving toward some point in the maze
    c) Finding cover
2) Shooting
    a) Sighting
    b) Opening fire
    c) Correction relative to recoil and motor reflexes
3) Visual recognition
    a) Recognition of objects
    b) Recognition of dangerous zones
    c) Recognition of enemy

The simplest set of rules may be the following:

- $1a) \wedge 1c) \rightarrow 1a)$ (just bypassing)
- $1a) \wedge 3b) \rightarrow 2a) \wedge 1b)$ (accurate obstacle avoidance)
- $1a) \wedge 3c) \rightarrow 2a) \wedge 2b) \wedge 2c)$ (shooting after enemy recognition)

We give only a superficial representation of these rules, which are much more complicated when we go into details with respect to implementation in game engine.

Rule based system allow us to implement the neural network method when we have to choose another rule with the same premise, for example, if we have a rule:

$$1a) \wedge 3c) \rightarrow 1b) \wedge 1c) \quad (2)$$

This rule should be used, if a player has low health and his chance to win against enemy is almost zero, so he should hide and create an ambush for an enemy.

If we have time-limit and command should defend some place we should use the following rule:

$$1a) \wedge 3c) \rightarrow 1b) \wedge 2b) \quad (3)$$

So we will distract an enemy or sacrifice player to win some time for a command.

In a real game the player chooses different rules depending on his psychological temper, emotions or calculations, so it is really a good place to use learning techniques.

For each type of game we produce rules, which say whether or not player achieve his current goal and increase a part of command goal for this game. Of course command goal has greater priority, but in fact, in each situation the player decides without hesitation operating with current situation and parameters from visual recognition.

Formal Concept Analysis methods are used to find out a basis of rules (implications) when we have a non-empty intersection of premises. Machine Learning is used to find weight combination for rules, which have the same premises.

Model of neuron networks can be applied to univariate and multivariate analysis. In our work using neuron approach is appropriate to imitate personal and team player's goals. One player worries about its level of health; also, it is interested in winning. These two ideas characterize its behavior during the game; it means that the following situation is possible: BOT may decide to shoot and be with the low level of its health, because it will win. But, when we consider team game, BOT will not risk by it-self, it will defend, possible, shoot or run away. This case of game is complicated, but we can divide it into several simple cases, each of them is the same as for personal game. That is why neural networks can successfully reflect behavior of players in shooter.

6 Process of Gaming

Process of gaming starts with visual recognition. As far as BOT defines an enemy with certain threshold probability or after enemy fire identification, it starts to aim. Given that X is the horizontal rotation change speed, and Y is the vertical. The relative error depends on the angle between BOT's rotation movements and direction on the target, and the sign of this angle. It also depends on X-speed and Y-speed: the stronger the need to change the speed of rotation change, the greater the relative error will occur.

Let's consider two situations of possible targeting. In both examples visual recognition shows the place of the enemy. Also, we can see the rotation of BOT, which describes by means of the sum vector of X and Y. Black circles shows the approximate trajectory of aiming which is close to the human response to the detection of the adversary. The figure 5 represents the situation in which the direction to the target is opposite to the BOT rotation, therefore the relative error will be very high.

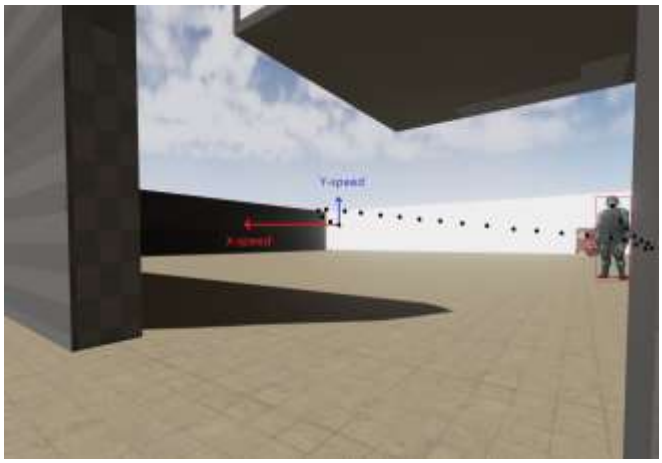

Fig. 5. Example 1

The figure 6 displays the case in which the direction to the goal and the rotation are the same; consequently, the relative error will be rather low.

Fig. 6. Example 2

So, we divide aiming by three phases:

- inertia during recognition,
- instant aiming,
- braking or correction near the goal,

and include them in our model.

7 Current Progress and Conclusions

We have stated game AI model and test it manually on demorecords. We will provide statistical parameters from experiment on one enemy at three-dimensional maze with over 500 different positions of players. In the current state we are verifying the parameters to identify the dangerous zones and we proceed with the comparison of the methods of visual recognition to apply for our model. RBS is now constructed automatically using Unreal Engine 4 and we recalculate bases of implications for every addition of new rules block. The system is flexible enough to extend to a new rule and to recalculate parameters of neural networks for a problem of rule choice with similar premises.

The following picture represents the current work of our AI:

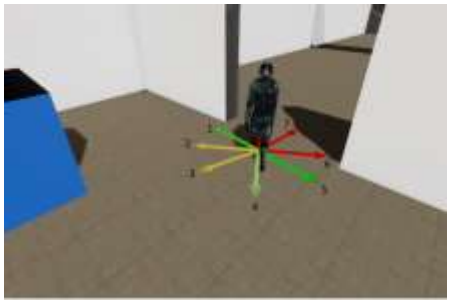

Fig. 7. Looking around the corner

The BOT AI can decide in which direction it will move on. Bright green arrows (1, 5) show that appropriate routes are the most probable, whereas red arrows (6, 7) correspond to the least probable paths. Yellow cursors (2, 3) display that such directions are possible, but the probability is not very high.

We aim to implement the blueprint, which will correspond to target sighting with adjustments. At the current progress we test the model of targeting with respect to dedicated case of one moving enemy at scene, but we will continue to verify this module with many enemies and partial recognition cases after completing results on the best visual recognition models.

Acknowledgements

The authors would like to thank the colleagues from Higher School of Economics Geoffrey Decrouez for great support during preparation process and Anton Konushin for well-timed advice on visual recognition methods.

References

1. Yanovich I. “Review of Call of Duty: Ghosts” (in Russian) http://kanobu.ru/articles/syuzhetnaya-kampaniya-call-of-duty-ghosts-367451/
2. Yang G. and Huang T. S., *“Human Face Detection in ComplexBackground”*. Pattern Recognition, vol. 27, no. 1, pp. 53-63, 1994.
3. Kotropoulos C. and Pitas I., “*Rule-Based Face Detection in Frontal Views”.* Proc. Int'l Conf. Acoustics, Speech and Signal Processing, vol. 4, pp. 2537-2540, 1997.
4. Pham T. V., Worring M., and Smeulders A. W. M. “*Face detection by aggregated bayesian network classifiers”*. Pattern Recognition Letters, 23(4):451-461, February 2002.
5. Henry A. Rowley, Shumeet Baluja, and Takeo Kanade. “*Human face detection in visual scenes”*. Technical Report CMU- CS-95-158R, School of Computer Science, Carnegie Mellon University, November 1995.
6. Roth D., Yang M.-H., and Ahuja N. “*A SNoW-based face detector”.* Advances in Neural Information Processing Systems 12 (NIPS 12), MIT Press, Cambridge, MA, pp. 855-861, 2000.
7. Viola P. and Jones M. “*Robust Real-time Object Detection”.* In Proc. 2nd Int'l Workshop on Statistical and Computational Theories of Vision - Modeling, Learning, Computing and Sampling, Vancouver, Canada, July 2001.
8. Vorontsov K.V. “Bayesian classification algorithms. Drafts of lectures” (in Russian) http://www.ccas.ru/voron/download/Bayes.pdf
9. Heckerman, D. *“A Tutorial on Learning with Bayesian Networks”.* In Jordan, M. (Ed.), Learning in Graphical Models, MIT Press, 1998.
10. Vezhnevec A. “ Popular neural network architecture” (in Russian) http://cgm.graphicon.ru/metodyi/populyarnyie_neyrosetevyie_arh itekturyi.html
11. Vapnik V. and Lerner A. J. “*Generalized portrait method for pattern recognition”.* Automation and Remote Control, vol. 24, no. 6, 1963.
12. Roth D. “*The SNoW Learning Architecture”.* Technical Report UIUCDCS-R-99-2102, UIUC Computer Science Department, 1999.
13. Ming-Hsuan Yang, David Kriegman, and Narendra Ahuja. “*Detecting Faces in Images: A Survey”.* IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), vol. 24, no. 1, pp. 34-58, 2002.
14. Viola P. and Jones M. *“Robust Real-time Object Detection”.* In Proc. 2nd Int'l Workshop on Statistical and Computational Theories of Vision - Modeling, Learning, Computing and Sampling, Vancouver, Canada, July 2001.
15. Szelinski R. “Computer Vision: Algorithms and Applications”. Springer, 2011.
16. https://www.unrealengine.com
17. http://unreal-engine4.ru
18. Champandard A.J. “AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors”. New Riders Games, 2003.