ECONTECHMOD. AN INTERNATIONAL QUARTERLY JOURNAL – 2018, Vol. 07, No. 4, 21 – 28

Designing multi-agent gameplay

V. Teslyuk, A. Lozunskyi, T. Teslyuk, P. Denysyuk

CAD Department, Lviv Polytechnic National University, UKRAINE, Lviv, S. Bandery Str, 12
e-mail: vasyl.teslyuk@lpnu.ua, lozunskuj.andrij@gmail.com,
taras.teslyuk@gmail.com, pavlo.denysyuk@gmail.com

Received December 01.2018: accepted December 18.2018

Abstract - There is an attention shift within the gaming industry toward the more natural behavior of nonplaying characters. The first part of this paper says about the main concepts of agent and multi-agent systems. The main part of this paper describes the main problem in any multi-agent system, agent communication. Then we write about main concepts of designing multi-agent gameplay and describes own developed MAS game simulator.

Keywords : MAS, agents, ACM, JADE, gamebot.

INTRODUCTION

Over the past ten years, the development of telecommunication technologies has led to creating new concepts of distributed and intelligent software systems. Such systems are implemented in different ways, but multi-agent systems (MAS) concentrate all the properties which are necessary for technologies which are needed to design and develop simulation systems and game systems with proper artificial intelligence [1-3, 21-23]. The results from the introduction of agent technologies confirm the promise of this direction. Technology and the theory of agents continue to evolve through research and commercial projects. Particular attention is paid to the integration of methods of artificial intelligence [3], which until now have been used mainly in studies. Systems with decentralized management are built on the basis of autonomous, environmentally adaptable agents (active elements) that achieve systemic goals through local information exchange and coordination [1, 17-20, 24-26].

The more complex the games become and the more elaborate the interactions between the characters during the game, the more difficult it will become to design these characters without the use of specialized tools geared toward implementing intelligent agents in a modular way [6, 14, 15]. The main issue of this paper is thus, given that it makes sense to use ideas from agent research in gaming, as seems to be supported by the growing amount of work in games that incorporates agent concepts and technologies, what would be necessary to make use of the agent technology as developed for the multi-agent platforms.

BASIC CONCEPTS AND PROPERTIES OF THE AGENT

There is no clear definition of the term of the agent. All available definitions are based on the scope of agents. Such scopes are artificial intelligence, software development, the field of study, computer science, engineering, etc. [4]. Each agent is a process that has a sufficient amount of knowledge about the object and the ability to share this knowledge with other agents. In terms of the object-oriented approach, an agent can be considered as a set of functions in combination with an interface that is capable of sending replies and receiving information. You can also define the agent as a computer program that executes asynchronously in accordance with the behavior of a particular person or organization. Today, there are two main types of agents: stationary and mobile [8]. An agent can be perceived as a constituent system that responds to the environment through the sensors and acts on it through the effector [1]. According to this definition, the agent is any entity (physical or virtual) that is experiencing its environment and performs certain actions.

Physical are called agents that directly through the controllers are in contact with the external environment and parts of its own system [5].

Virtual agents are agents who, through software that only receives data from the sensors, sends data to the server and, when it received, gives the tasks to the physical agent [6].

In most cases, the agents are the combination of physical and virtual components. Using this information, we can formulate a more proper definition: "Autonomous agents are computing systems included in some complex dynamic system that contact the environment through sensors, the senders, sending data to the server through a virtual component and following the instructions received" [5]. Figure 1 depicts the scheme of the agent's operation.

In addition to touch inputs, actions, and goals, the agent may include domain knowledge (knowledge of a specific environment or issues to be addressed). This knowledge can be algorithmic, based on methods of

artificial intelligence, heuristics, etc. The notion of the environment includes a physical system, operating system, internet, or perhaps some of these systems are combined.

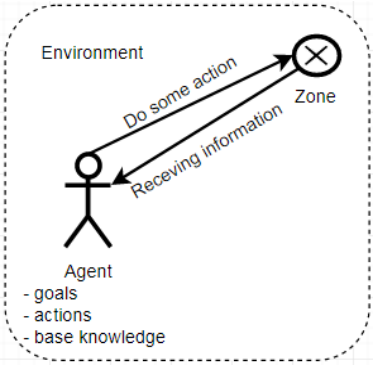

Fig.1 Scheme of the agent's operation

If the agent responds simultaneously to changes in the environment and converts received sensory data into a certain action, this agent is known as reactive (sometimes called reflex agent). Usually, reactive agents [1, 3] do not store the internal state, after receiving any data this agent immediately affects the environment. On the other hand, if the agent maintains the internal state and implies the consequences of his actions, this agent is called the trained agent [1]. Figure 2 depicts the functional scheme of the reactive agent, and in Figure 3 depicts the functional scheme of the agent having the property of learning. The first step in developing a multi-agent system is to select the properties of the agents.

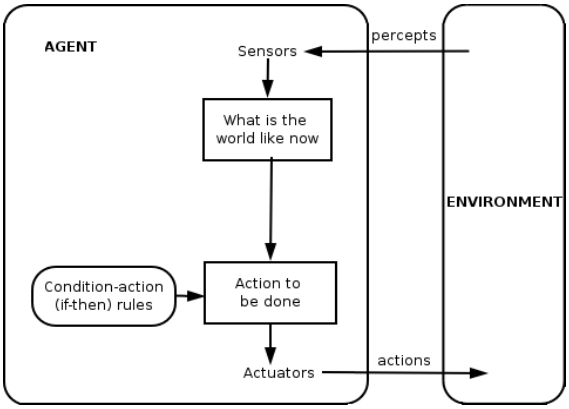

Fig.2 Functional scheme of the reactive agent, https://en.wikipedia.org/wiki/Multi-agent_system

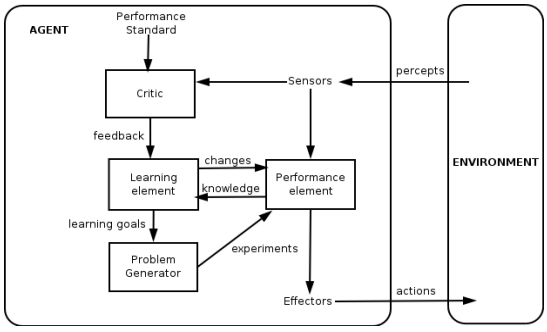

Fig.3 Functional scheme of the trained agent, https://en.wikipedia.org/wiki/Multi-agent_system

THE MAIN PROPERTIES AND CLASSIFICATION OF MULTI-AGENT SYSTEMS

Multi-agent systems is a powerful tool for solving tasks that are impossible or not profitable with the help of one agent. Multi-agent systems have the following properties [7]:

1. Autonomy - the ability when agents can resolve their tasks without human intervention.

2. Social - the ability to live in the community of agents and exchange data with other agents.

3. Reactivity - the ability to perceive the state of the environment and respond in a timely manner to changes in a given environment.

4. Internal activity - the ability to take the initiative, that is, not only react to external events, but also generate goals and act rationally for their achievement.

The fact that agents within the MAS work together means that such cooperation is individual for each system. However, the concept of cooperation in multi-agent systems is the most complex part of all MAS and it is more problematic than in the case of agents, which makes data systems complex and individual.

There is an infinite number of simulators, games, systems that are designed with the basic principles of multi-agent systems. All systems have their advantages and disadvantages, therefore it is customary to classify them according to certain properties and tasks facing the multi-agent system. Multi-agent systems are classified into [9]:

1. Discretely independent multi-agent systems.

2. Independent MAS without cooperation.

3. Cooperative multi-agent systems.

Discretely independent multi-agent systems are found in environments that enable the task to be solved without the communication of agents. As an example of this type, we can use two agents (controllers) that control the synchronous generator in the power grid: the automatic voltage regulator (AVR) and the speed controller. They have different goals that are not related to each other [10].

Independent MAS without cooperation are developed independently and each individual agent pursues his or her own goals independently of others. It is important to emphasize that in these multi-agent systems, agents are not aware of the existence of other agents, and each agent considers others as part of the environment. Since agents exist in the same environment and perceive each other as unknown environments, agents can accomplish one goal without knowing about collaboration, but they can also interfere with each other's tasks. Collaboration between independent agents may occur in two ways [11]: 1) Individual agents may receive information about another agent as sensory data. 2) Separate agents can receive as touch data information about the task performed by an unknown agent and thus

perform another task. This type of multi-agent systems is used in systems where agents perform the same tasks.

Cooperative MAS is the systems that know about other agents in the system and this agent communicate with others in various ways [8].

AGENT COMMUNICATION

In multi-agent systems that consist of many autonomous agents, negotiations are a key form of interaction that allows agent groups to reach a reciprocal agreement, for example, with respect to some idea, purpose, or plan. Communication between agents can be achieved in a similar fashion as actions and perception of the agent. Sending a message consequently requires describing the pre- and postconditions. There are three main topics for negotiating research [6]:

Protocols of negotiations - a set of rules that govern the interaction. They determine the permissible types of participants, negotiate, events causing transitions to other states and acceptable actions of participants in certain states.

Negotiations are the range of issues that should be agreed upon. These can be, as simple problems, as complex problems [11]. Permissive operations on these subjects are also relevant. In the simplest case, the structure and content of the agreement are established and the negotiations seek to accept or reject the proposal. A more complex case involves flexibility in order to change the significance of the problem in the negotiation, through counteroffers, changing the structure of the subject of negotiations.

Models of consideration of agents - provides a decision-making machine through which the participants try to achieve their goal. The complexity of the model is determined according to the protocol used, the nature of the subject of negotiations and the range of operations that can be performed on it.

Negotiations in the MAS may contain complex considerations of a high level. In order to create a practical use, a low-level basic infrastructure is very important. For example, any formally-based negotiation protocol should deal with a loss of communications or the elimination of one of the negotiating partners.

Indeed, low-level communication languages control classes of conversations in which pair of agents can participate. The components of conversations are determined not only by the indicated action but also according to the preconditions, postures, and conditions of the completion of this speech act [5]. They determine the circumstances under which communication may occur and include the status of the submission of the agents concerned.

In order for the negotiations to be completed successfully, all involved parties should clearly understand the rules of the protocol negotiations or commitments. Indeed, low-level communication languages control classes of conversations in which pair of agents can participate. The components of conversations are determined not only by the indicated action but also according to preconditions, post-words, and conditions of the completion of this speech act. They determine the circumstances under which communication may occur and include the status of the submission of the agents concerned.

Receiving a message is controlled by an agent subscribing to messages and by the game engine when it determines that an agent can sense an action. In this case, we do not only specify the agreement between agent and game engine but also between agents. So there are three or more parties that need to conform to the agreement instead of two in the previous case. We will call this agreement the agent communication model (ACM) [4].

The definition of the ACM will contain the type of things that can be communicated (communication content representation) between agents. This representation is only relevant to the agents in the game. The ACM also specifies when communication can take place (qualification representation). This specifies the impact of the environment on communication. For instance, if an agent is far away, it may not be able to communicate. These factors are relevant to the game engine that is responsible for simulating the environment.

There already exists a formalism that provides a communication content representation. It provides a way to communicate such things as beliefs among agents or propose an action or communicate with multiple agents. Within the FIPA standard [11], these communicative acts are already defined. We propose to use the FIPA standard to establish a game-specific message structure. For example, in our firefighting game, agent A may propose to agent B that A opens the door to the building:

```
(propose
   : sender
      (agent-identifier:name A)
   :receiver
      (set (agent-identifier:name B))
   :content
      "((action A (open door))"
   : ontology Fire-fighting
   :In-reply-to proposal2
   : language Fipa-so)
```

The message is translated to the concepts internal to both the agent and the game engine. The game engine will, upon reception of the above message, send this message to agent B and automatically subscribe agent A to the communication messages of agent B. This is because agent A and B can now be said to be in a dialogue and it is likely that agent A would like to receive an answer. The game could progress such that agent B replies affirmatively and the game engine receives an action from agent A to open the door and an action from agent B to go through the door. The game engine will now have enough information to know that this is a

coordinated action and that the order of actions (as implied by the dialogue) is to process the door opening action first and the movement action second. The game engine, therefore, takes these messages from the incoming actions queue and processes these together (coordinated) and in the right order [12].

To describe the impact of the environment on communication, we have to augment the linguistic representation with information about the environment. Since communication is a form of action, the same qualification representation needs to be made explicit. These qualification rules will also need to specify the ramifications of communication. This allows us, on the one hand, to specify what is needed when agents want to communicate (e.g., that they are close together) and on the other hand the (side) effects of communication (e.g., if other agents than the message recipient are nearby they too may receive the message):

```
Get: Poss(Send(Propose(Action,Agent)))
        Dist(Agent)<5
POST: Done(Send(Propose(Action,Agent)))
        Dist(Agent')<5
        Poss(Receive(Propose(Action,Agent)))
```

DESIGNING GAME SIMULATION

The designing game simulation consists of two main part: Client-side, Server-side.

Client-side(gamebots) has been created as a research platform for making the connection between agent research and a computer game, namely, the Unreal Tournament environment, and is one of the most used client-side implementations. In client-side approaches, agents are running as completely separate programs from the server and are usually communicating through network sockets. Network communication between agents and other external software programs has been successfully used in other multi-agent systems. Gamebots was designed for educational purposes, and therefore, multiple client implementations have been created, for example, one using the scripting language TCL, a SOAR bot, and a JAVA-based implementation [7].

Figure 4 shows a diagram of the different Gamebot modules in combination with the JAVAbot extension. The Gamebot API forms the extension to Unreal Tournament that is needed to connect a client-side program to the Unreal Tournament. The JAVAbot API is the client side of the coupling. Having a general JAVA API on this side facilitates the connection to most agent platforms because they are usually also JAVA based. Information is sent from the game engine to the agents through the Gamebot and JAVAbot APIs by two types of messages: synchronous and asynchronous messages [6]. The synchronous messages are sent at a configurable interval. They provide information about the perceptions of the bot in the game world and a status report of the bot itself. Asynchronous messages are used for events in the game that occur less often and are directly sent to the agent.

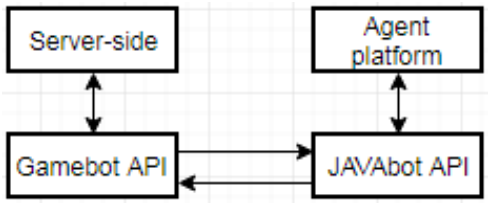

Fig.4 Client-side bots architecture in game design [7]

The Gamebots is actually not a pure client-side solution because the server is also modified to supply a special world representation to the bot. There are some pure client agent implementations, but they are usually only created for cheating purposes. In this case, the processing of the data is done in the bot itself because it pretends to be a human client game. Doing this filtering on the server is more efficient because only the useful information needs to be communicated. This server modification, the Gamebots network API, performs a similar task and for similar reasons as the area awareness system in game simulation. This clarifies why the Gamebot API is specific for Unreal Tournament; it needs to know the internal representation of the game world in order to make the translation (efficiently) [12].

The Gamebot API does not provide information about the complete environment, but only about objects that are perceivable by the bot. Thus, if a bot wants to gather information about the complete environment, it has to (physically) explore it. To navigate, for example, the agent receives information about predefined navigation nodes in the game map, but only the currently observable nodes are returned. The agent thus does not know what exists around the corner, let alone that it can reason about it. Due to the representation choices made in Gamebots, information about the environment has to be stored at the agent side of the system [11]. This results in large differences between the agent's representation of the environment and the actual environment of the game engine. For complex bots, the information provided by the Gamebot API quickly becomes too limited to make intelligent decisions. For example, the agent cannot know the spawning location of a certain power-up, and therefore, it cannot plan to go there.

The Gamebot API does not provide information about the complete environment, but only about objects that are perceivable by the bot. Thus, if a bot wants to gather information about the complete environment, it has to (physically) explore it. To navigate, for example, the agent receives information about predefined navigation nodes in the game map, but only the currently observable nodes are returned. The agent thus does not know what exists around the corner, let alone that it can reason about it. Due to the representation choices made in Gamebots, information about the environment has to be stored at the agent side of the system. This results in large differences between the agent's representation of the environment and the actual environment of the game engine. For complex bots, the information provided by the Gamebot API quickly becomes too limited to make intelligent decisions.

For example, the agent cannot know the spawning location of a certain power-up, and therefore, it cannot plan to go there [9].

In the most games, the agents are completely integrated in the default game loop in the same way as the physics engine, the animation engine, and rendering engine. The agent's decisions are defined by a sequence of method calls, and the methods return the action that has to be performed at that time step. Direct method calls can be used for many different decision-making processes, for example, hard-coding approaches, directly specifying what to return with a certain input; fuzzy logic, mapping the right output to a certain set of input variables; or finite state machines, identifying the situation the agent is in and executing the corresponding method call. Independent of the particular decision-making strategy, the whole process is completely synchronized [1]. This limits the complexity of behavior because in a synchronized process all decisions have to be made within one-time step, and complex decisions would slow down a game too much.

Figure 5 gives an impression of the implementation of agents in games simulation. On the lowest level in the figure, a translation from the raw engine data to a representation more suitable for agents has been created, called the area awareness system (AAS). The heart of the AAS is a special 3D representation of the game world that provides all the information relevant to the bot. The AAS informs the bot about the current state of the world, including information relevant to navigation, routing, and other entities in the game. The information is formatted and preprocessed for fast and easy access and usage by the bot. For instance, to navigate the bot receives information from the AAS about the locations of all static items, and it can ask the AAS whether a certain location is reachable [7]. The AAS is responsible for route planning. The first level also executes the actual actions of the agent and facilitates the decision process of the agents. However, the agents are highly dependent on the data they can extract from the AAS, for example, an agent cannot decide to take another route to a certain item. To illustrate the importance of the linkage between the engine and the agents, this part constitutes over 50% of the entire agent code.

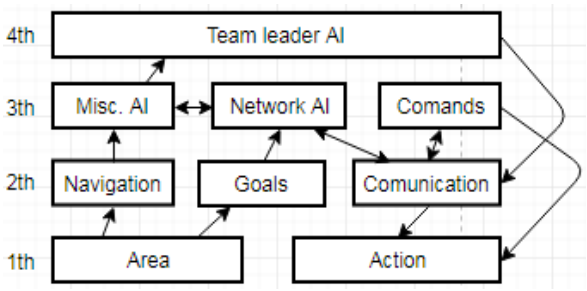

Fig.5 Bot`s server-side architecture in the game design

On the second and third levels of the architecture, the information from the AAS can be used to check whether the bot's goals are reached or how far off they are. Depending on the character that a bot plays, the fuzzy logic control determines which of the possible paths the bot should start navigating.

Little communication between agents takes place in a normal game. It is only used to assign roles in team play situations [5]. Communication is implemented by using the chat system for sending simple text messages. More cooperation between agents would require improved communication facilities. Moreover, currently, it is assumed that communication is always successful, which is usually not guaranteed in realistic mult-iagent scenarios.

IMPLEMENTATION GAME SIMULATION

Based on the main concepts of game designing, there was developed game simulation of "Bacterial War". Several variants of agents, each of which has their reaction time and internal algorithm for analysis of the situation, are implemented in the work. Figure 6 is depicted as a general scheme of agents functions.

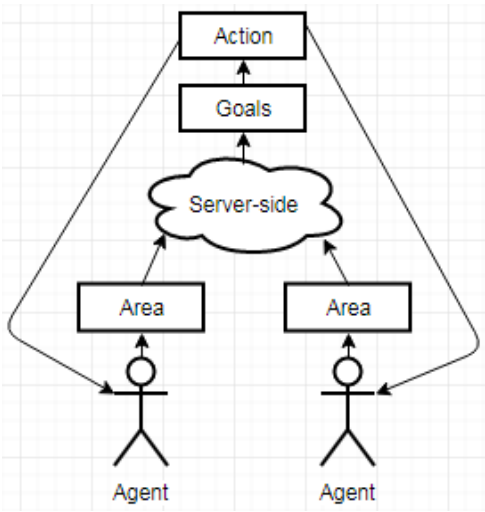

Fig.6 Scheme of agent's functionality

The agent is responsible for choosing the direction of movement of groups of bacteria from a given colony. At one time, the colony can choose no more than one direction of movement. In case of loss or capture of the colony, the agent changes his subordination to the player. The reaction time determines how often an agent can be involved, analyze the situation and choose a direction. There are three main approaches to implementing actions that are implemented in the agents:

1. Analysis and search of opponents, opponent's choice. Designing a relocation route to an opponent if it is achievable (developing a capture strategy).
2. Analyze and search agents, select the agent who needs help. Designing a relocation route to an agent if it is achievable.
3. Analysis of projected routes and solution correction.

The analysis and search of the opponent take place using the basic ideas of the algorithm A *. Algorithm A * will first visit those vertices that are likely to lead to the shortest path. In order to recognize such executions, each

well-known top of x equals the value of f (x), which is equal to the length of the shortest path from the original vertex to the finite, which runs through the selected vertex. The vertices with the smallest value f are chosen first.

The function f (x) for the vertex x is defined as:

$$f(x) = g(x) + h(x), \quad (1)$$

where: *g (x)* is a function whose value is equal to the cost of the path from the original vertex to *h(x)*, *h (x)* is a heuristic function that evaluates the cost of the path from the vertex *x* to the finite.

In the process of the game design, the following algorithm was used:

Step 1. After selecting the level of the game, the game field is initialized, the initial display and initialization of the agents.

Step 2. After the command about the beginning of the game, the game controller is started. He is responsible for controlling gaming mechanics, deducting current and internal time, displaying the playing field and launching agents.

Step 3. On each of the streams, the condition of each of the colonies and the group of bacteria and agents is checked.

Step 4. If the colony has a simulated route to other colonies, a group of bacteria is formed and sent on a given road.

Step 5. If a group of bacteria reaches the colony of the enemy, it causes damage to the colony. Otherwise, it strengthens its own colony.

Step 6. If the agent is free, he analyzes the situation: he evaluates his opponents and neighbors and makes a decision on laying the road to improve the game situation (develops a strategy).

Step 7. At the end of the stream, the balance is checked and the condition is checked at the end of the game.

Step 8. Draw the location of the colonies, roads, and groups of bacteria on the game screen.

Step 9. If the game is not completed, go to step 3.

In practice, it is rather difficult to investigate multi-agent systems, since it is virtually impossible to predict the actions of agents, as it is impossible to predict ways of communication between agents. For a more realistic study of systems, use simulation models that directly reflect the behavior of agents in the system.

The simulation model is constructed in the form of a strategic game program that includes: bacteria that form colonies that are looking for the most optimal ways to capture all vertices (planets) and the game field.

Each of the bots analyzes the situation on the field and synthesizes its solution, which would be beneficial to capture the next vertex, that is, what strategy should be built for further colonization of the playing field and how to strengthen its presence on already conquered peaks. Figure 7 shows the main menu of the simulation game Bacterial War [13, 16].

Fig.7 Main menu of the imitation game

In addition, interactive collaboration between hostile colonies that can be combined to fight a stronger opponent or support an attack at a time of the joint attack is implemented in the work.

Figure 8 shows an example of the functioning of agents in the game simulation system. This example shows a system with three colonies of different colors. Each of the colonies - this system is endowed with elements of artificial intelligence; whose purpose is to capture the entire field. Field capture occurs by sending their agents to the enemy colony to increase the influence of their bacteria in a given colony.

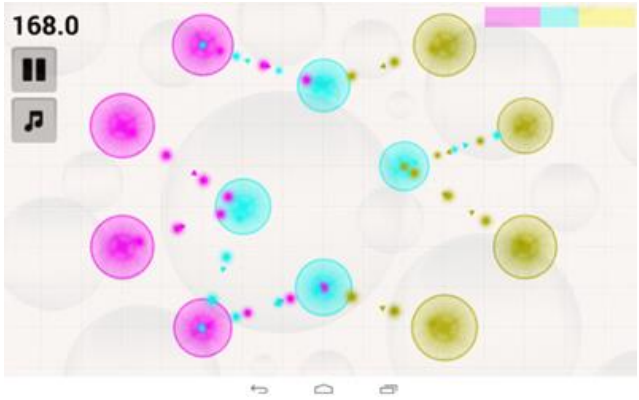

Fig.8 Example of agent functionality

CONCLUSION

There is the consensus among game developers that intelligent characters for games can make games better. However, there is a difference in the approach to bring intelligence about between the game developers and the artificial intelligence researchers. Consequently, using agent technology in combination with game technology is not trivial. Because agents are more or less autonomous they should run in their own thread and can only be loosely coupled to the game engine.

In this paper, there are arguments that improving the

AI in games by using agent technology to its full extent involves solving the issues above. Furthermore, solving the synchronization, information representation, and communication issues requires more than constructing a technical solution for the loose coupling of some asynchronous processes. Although this aspect is a fundamental part of the coupling, we also need to provide support on a conceptual and design level. Using a conceptual stance allows for connecting the agent concepts to the game concepts such that agent actions can be connected to actions that can be executed through the game engine and that agents can reason intelligently on the information available from the game engine.

An imitation model based on the multi-agent decision-making approach has been developed. The communication of agents is realized on the basis of the homogeneous MAS principle. The system is developed on the basis of an imitation model that mimics the actions of bacteria in the environment.

REFERENCES

1. **Ayesh A., Stokes J., and Edwards R., 2007.** "Fuzzy Individual Model (FIM) for realistic crowd simulation: preliminary results," in Proceedings of IEEE International Conference on Fuzzy Systems (FUZZ '07), pp. 1–5, London, UK.
2. **Boyko N., Kutyuk O. 2016.** Basic concepts of evolution in agents calculating and agents system. ECONTECHMOD. An International Quarterly Journal, Vol. 05, No. 2. 69-76.
3. **Wooldridge M., 2009.** An Introduction To Multiagent Systems. 468.
4. **Orkin J., 2003.** "Applying goal-oriented action planning to games," in AI Game Programming Wisdom 2, Charles River Media, Brookline, Mass, USA.
5. **Pollack M. E. and Horty J. F., 1999.** "There's more to life than making plans: plan management in dynamic, multiagent environments, "AI Magazine, vol. 20, no. 4, pp. 71–83.
6. **Lees M., Logan B., and Theodoropoulos G. K., 2006.** "Agents, games and HLA, "Simulation Modelling Practice and Theory, vol. 14, no. 6, pp. 752–767.
7. **Antunes L. and Takadama K., Eds., 2007.** Multi-Agent-Based Simulation VII, vol. 4442 of Lecture Notes in Computer Science, Springer, Berlin, Germany.
8. **Silverman B. G., Bharathy G., Johns M., Eidelson R. J., Smith T. E., and Nye B., 2007.** "Sociocultural games for training and analysis," IEEE Transactions on Systems, Man and Cybernetics, Part A, vol. 37, no. 6, pp. 1113–1130.
9. **Dastani M., 2008.** "2APL: a practical agent programming language," Autonomous Agents and Multi-Agent Systems, vol. 16, no. 3, pp. 214–248.
10. **Bordini R., Hübner J., and Wooldridge M., 2007.** Programming Multi-Agent Systems in AgentSpeak Using Jason, John Wiley & Sons, New York, NY, USA.
11. **Kraus S., 1997.** "Negotiation and cooperation in multi-agent environments," Artificial Intelligence, vol. 94, no. 1-2, pp. 79–97.
12. **Johnson W. L., 1994.** "Agents that learn to explain themselves," in Proceedings of the 12th National Conference on Artificial Intelligence, vol. 2, pp. 1257–1263, Seattle, Wash, USA.
13. **Teslyuk T., Denysyuk P., Kernytskyy A., Teslyuk V., 2015.** "Automated Control System for Arduino and Android Based Intelligent Greenhouse" in Proceeding of the XIth International Conference "Perspective Technologies and Methods in MEMS Design", MEMSTECH'2015, Polyana, Lviv, Ukraine. 2015. – P. 7 – 10.
14. **Chernyshev Y., Chumachenko D., Tovstik A., 2013.** Development of intelligent agents for simulation of hepatitis B epidemic process. Proceedings of East West Fuzzy Colloquium (20th Zittau Fuzzy Colloquium, September 25 – 27, 2013). 161 – 168.
15. **Chumachenko D., Yakovlev S. 2016.** Investigation of agent-based simulation of malicious software. ECONTECHMOD. An International Quarterly Journal, Vol. 05, No. 4. 61-67.
16. **Teslyuk, V. M., Lozynskyi, A. Ya., & Teslyuk, T. V. 2017.** Application of Multi-Agent Approach in the Process of Implementation of the Game Program "Bacterial War". Scientific Bulletin of UNFU, 27(6), 178–181. (In Ukrainian)
17. **Nwana H., 1996.** «Software agents: An overview». – The Knowledge Engineering Review Journal – vol. 11, № 3 – pp. 1– 40.
18. **Carole B., Marie-Pierre G., Sylvain P., Gauthier P. 2003.** ADELFE: A Methodology for Adaptive Multi-agent Systems Engineering – Engineering Societies in the Agents World III, Lecture Notes in Computer Science – Volume 2577 – pp. 156 – 169.
19. **Lytvyn V., Dosyn D., Medykovskyj M., Shakhovska N. 2011.** Intelligent agent on the basis of adaptive ontologies construction – Signal Modelling Control – Lodz.
20. **Kubera Y. 2010.** Everything can be Agent (Extended Abstract) – Proc. of 9th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010). – Toronto – Volume 1 – pp. 1547–1548.
21. **Boreiko O. Y., Teslyuk V. M., Zelinskyy A., Berezsky O., 2017.** Development of models and means of the server part of the system for passenger traffic registration of public transport in the "smart" city. Eastern-European Journal of Enterprise Technologies. – Vol. 1, Issue 2 (85). – P. 40–47.
22. **Teslyuk V., Beregovskyi V., Denysyuk P., Teslyuk T., Lozynskyi A., 2018.** Development and Implementation of the Technical Accident Prevention Subsystem for the Smart Home System, International

Journal of Intelligent Systems and Applications(IJISA), Vol.10, No.1, pp.1-8.

23. **Lytvyn V. 2013.** Design of intelligent decision support systems using ontological approach. ECONTECHMOD 2, (1), 31–37.
24. **Kravets P., 2018.** Game Method for Coalitions Formation in Multi-Agent Systems, 2018 IEEE 13th International Scientific and Technical Conference on Computer Sciences and Information Technologies (CSIT), Lviv, pp. 1-4.
25. **Burov Ye., Pasichnyk V., Katrenko A. 2018.** Building an ontology for system analysis. ECONTECHMOD 7, (3), 3–6.
26. **Lytvyn V., Oborska J., Vovnjanka R. 2015.** Approach to decision support Intelligent Systems development based on Ontologies. ECONTECHMOD 4, (4), 29–35.