JGOMAS 2.0: A Capture-the-Flag Game Using Jason Agents and Human Interaction

Luis Hernandez, Sergio Esparcia(✉), Vicente Julian, and Carlos Carrascosa

Departamento de Sistemas Informáticos y Computación (DSIC),
Universitat Politècnica de València, Camino de Vera s/n, Valencia, Spain
{lhernand,sesparcia,vinglada,carrasco}@dsic.upv.es

Abstract. Over the last few years educators have increasingly incorporated game simulations into higher education computer science curricula. Experiences have proved that students respond enthusiastically to these courses. According to this, this paper presents an evolved version of the JGOMAS simulator, which is a simulation game where students design and implement different types of agents and strategies in order to win the game. This new version allows students to practice different technologies related to the multi-agent paradigm as coordination, cooperation or decision-making.

Keywords: Multi-Agent Systems · Simulation games · Education

1 Introduction

The use of game design elements for teaching and learning computer science subjects has provided a number of advantages in recent years [1]. Developing computer games involves many aspects of computing, including computer graphics, artificial intelligence (AI), human-computer interaction, security, distributed programming, simulation... In this sense, there is a growing evidence of the effective use of games and game elements across Higher Education contexts and concretely in Computer Science degrees.

During different academic years, the toolkit JGOMAS 1.0 [2] has been used as a part of the learning process of students in a subject named Intelligent Systems of the Bachelor's Degree of Computer Science Engineering in the *Universitat Politècnica de València*. JGOMAS 1.0 is a simulation game where students can design and implement different types of agents and test different coordination strategies in order to win the game. Moreover, the toolkit allows the students to organize competitions between the developed teams. During the last years, the use of this toolkit notably improved the motivation of the students due to their interest in new areas such as behavior design, artificial intelligence, or the creation of multiplayer games. All these areas are covered by the JGOMAS toolkit.

Nevertheless, the first version of JGOMAS has different aspects that may be improved: (i) the user may observe the evolution of the game from different

J. Bajo et al. (Eds.): PAAMS 2016 Workshops, CCIS 616, pp. 173–184, 2016.
DOI: 10.1007/978-3-319-39387-2_15

points of view (several render engines may be attached to the same game), but cannot interact with the game to dynamically change its evolution. The experience can be dramatically improved including some kind of immersion of the students during the simulation; (ii) JGOMAS 1.0 is based on the JADE platform [3] and all their agents are implemented as JADE agents. This is not a problem by itself, but the use of any AI approach or sophisticated coordination techniques must be implemented from scratch. This problem can be solved by integrating existing, more complex, agent architectures into the toolkit; and (iii) current supported renders do not allow an optimal user experience since they do not support complex graphics.

Taking into account all these aspects, this paper presents an evolution of the JGOMAS toolkit. This new version allows students to work with heterogeneous teams of agents, being able to deal not only with JADE agents but also with JASON [4] agents. This new agent architecture allows students to create agents with a more sophisticated decision-making engine. Moreover, the toolkit has introduced the human in the execution cycle, so a student may interact with his team in a transparent way for the agents, which see him as other agent. Finally, a new render based on UNITY[1] has been also implemented.

The rest of the paper is structured as follows: Sect. 2 presents the background of the proposal; Sect. 3 introduces the new version of the JGOMAS toolkit; finally, some conclusions are commented in Sect. 4.

2 Background

This section describes the two works that are the background for the proposal of this paper. First, the Jason agent programming language is described. Second, JGOMAS, the framework for teaching agent programming, is described.

2.1 Jason

Jason [4] is an agent programming language based on AgentSpeak [5]. This language is based on the BDI (Beliefs-Desires-Intentions) architecture for defining cognitive software agents. In this approach, an agent is an entity composed of a set of beliefs, representing the current state of an agent and knowledge about the environment in which it is situated, a set of goals, which corresponds to tasks the agent has to perform/achieve, and a set of plans which are courses of actions, either internal or external, triggered by events, and that agents can dynamically compose, instantiate and execute to achieve goals. Jason extends AgentSpeak with different features, being the most important one the addition of speech-act based inter-agent communications. Other new features include: strong negation; handling of plan failures; support for developing Environments; support for MAS organizations and agents that reason about them; a library of essential "internal actions"; or an IDE in the form of a jEdit or Eclipse plugin.

[1] http://unity3d.com/.

2.2 JGOMAS

JGOMAS (Game Oriented Multi-Agent System based on JADE) [2] is a Multi-Agent System (MAS) for social simulation based on JADE [6] that has been developed to meet different purposes. Its main purpose is to serve as a starting point for studying the feasibility of the integration between Multi-Agent Systems (MAS) and Computer Graphics, including applications such as Virtual Reality.

Thus, JGOMAS adds a 3D rendering engine that depicts in real-time the execution of the MAS running in the background. This can be used to qualitatively evaluate and validate the implementation of the MAS.

As a test for the developed MAS, a capture-the-flag (CTF) game has been developed. In this kind of games, two teams compete against each other. Each team has a base, with a flag located inside it. The objective of the teams is to capture the flag from the base of the opponent and to bring it to its own base to score a point or to win the game. This type of game was firstly introduced in 1984 by the game *Bannercatch*, and is now a standard multiplayer mode for the first- and third-person shooters since it was made popular when it was first introduced as a modification to the *Quake* videogame. It is intuitive to implement a CTF game using a MAS because a player (i.e. a member of the team) can be modeled as an agent. Moreover, it is also interesting for studying agent cooperation, since each player has to collaborate with his teammates to achieve the objective of the game.

However, the JGOMAS version of CTF is slightly different. The two represented teams are the Allies and the Axis, each one having a base, but only the Axis have a flag. The objective of the Allies is to reach the Axis base, capture the flag and get it to their own base. If they reach the base within the established game limit, the Allied team wins. The objective of the Axis is to defend their base from the Allies. If the time expires and the Allies did not captured the flag, the Axis wins the game.

A domain like CTF enables a simple and entertaining way to establish a testbed either for algorithms and optimizations individually on each agent and for competitive and cooperative strategies, within and between teams.

After working with this first version of JGOMAS in both research and education domains, some aspects have been detected to be improved. These improvements will enhance the user experience and the possibilities offered by the second version of JGOMAS. The following section describes JGOMAS 2.0.

3 JGOMAS 2.0

3.1 Agents Description

As has been stated in the previous section, JGOMAS is primarily composed of two subsystems. On the one hand, there is a multi-agent system with two different kinds of agents. One of these types of agents handles the game logic, while the others belong to one of the two teams, and play the game.

Actually, this subsystem is a layer that runs on top of a multi-agent platform (specifically, JADE) and can take advantage of all the services provided by JADE.

JGOMAS 2.0, which is the current version of the JGOMAS platform allows the use of Jason agents to form teams of agents.

On the other hand, a new ad-hoc Render Engine to display a 3D virtual environment has been developed. According to the specific requirements of graphic applications (e.g., high computational cost for short periods), this engine has been designed as an external module (not as an agent). It has been written in UNITY 3D (Fig. 1 shows an example of a JGOMAS game displayed with this new render engine).

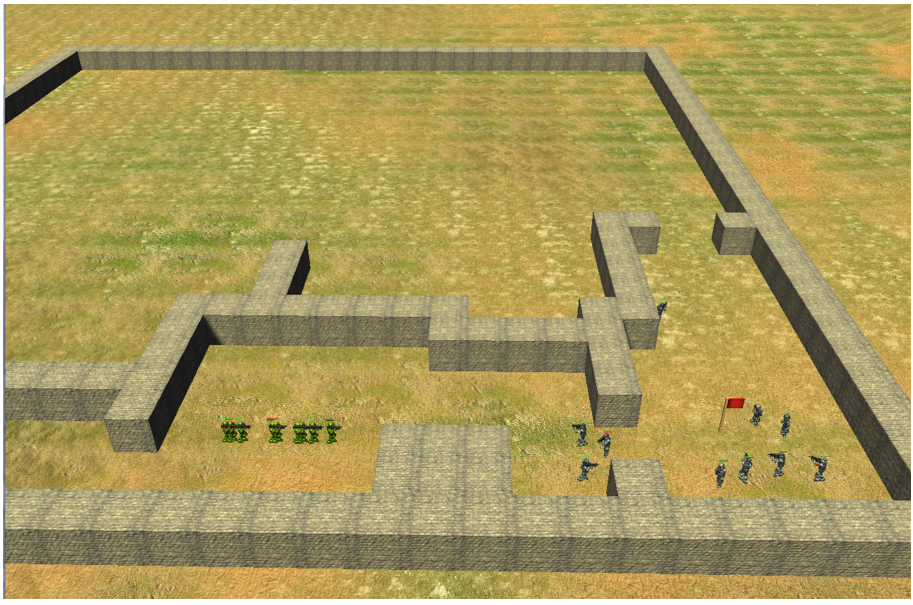

Fig. 1. JGOMAS render engine view

Figure 2 shows a JGOMAS architecture where all components and their relationships can be seen: the JADE platform as support for JGOMAS Multi-Agent System, which is comprised of agents, one of them acting as a controller for other agents, and as an interface for the Render Engine.

The Multi-Agent System JGOMAS can be viewed as a kernel (basic package), which provides an interface for the Render Engine to connect to the current game.

Agent Taxonomy. The following taxonomy of agents within JGOMAS according to their functionality may be established:

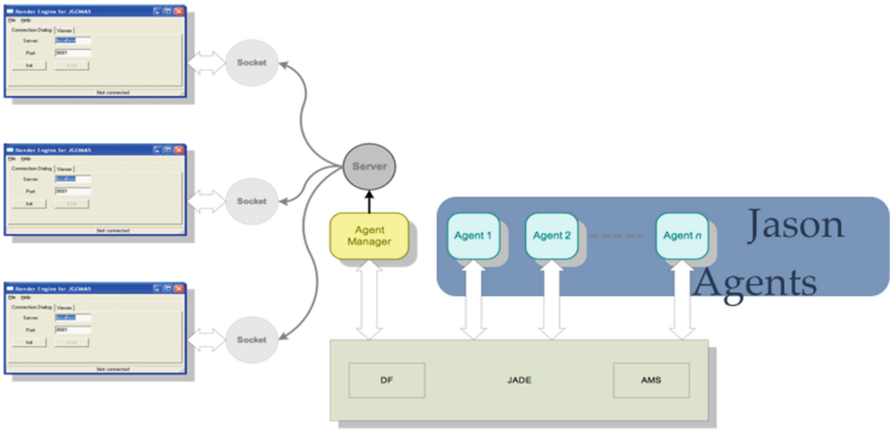

Fig. 2. JGOMAS 2.0 architecture

- Internal agents: are those which form themselves the JGOMAS management platform. Their behaviors are predefined, and the user cannot change them. These agents are JADE agents, and there exist the following types:
  - Manager: This is a special agent. Its main objective is to coordinate the current game. In addition, it answers requests from other agents. Another task that it is responsible for is to connect the game with the Render Engine.
  - Pack: There are three different types of packs, the medic packs (used to give health to agents), ammo packs (used to give ammunition to agents) and the objective pack, i.e. the flag to capture. All these agents are dynamically created and destroyed with the exception of the flag (there exists only one flag throughout the game and cannot be destroyed).
- External agents: They are the players. They have a predefined set of basic behaviors that the user can modify or even add new ones. These agents are developed in JADE or Jason. An agent can play a unique role in the current game. There are three roles defined, but the user can define new, each providing a unique service. Thus, these agents (also known as troop agents), specialize in the following three roles:
  - Soldier: provides a backup service (the agent goes to help its teammates).
  - Medic: provides a medic service (the agent goes to give medic packs).
  - FieldOps: provides an ammo service (the agent goes to give ammo packs).

External agents are integrated into the virtual environment, allowing the interaction between them (through the perception of nearby peers and enemies) which can lead to cooperation and coordination with teammates. Also, since agents are located in this virtual environment, they must take into account the features of the terrain where they are located (mainly walls). All the communication that takes place between the agents in the platform is performed by passing messages according to protocols established by the FIPA ACL [7].

Maps. JGOMAS uses different maps to define the virtual environment. These maps, by default, are of size 256×256, so that the position of the agents is given by its coordinates (x, y, z), where x, z take values between 0 and 255, whilst, in the maps supplied, y is always equal to 0 (these maps do not have height). Each agent has partial access to the map where the game is played, since despite having access to static information about it, can only perceive objects that are at a certain distance (within the "cone of vision").

Tasks. A task is something that an agent has to perform in a particular position of the virtual environment. There are various types of tasks, according to the different actions that an agent can perform in the virtual environment, being the main ones:

- TASK_GIVE_MEDICPAKS: A medic must generate packets of medicine in a particular place (the position of the agent which requested it and which has agreed to go to give them).
- TASK_GIVE_AMMOPAKS: A fieldops must generate packets of ammo in a particular place (the position of the agent which requested it and which has agreed to go to give them).
- TASK_GIVE_BACKUP: A soldier should go to help a teammate to a particular place (the position of the agent which requested it and which has agreed to give it go).
- TASK_GET_OBJECTIVE: The agent, from the ALLIED team, must go to the starting position of the flag for it. If it manages to grab the flag, this task becomes going back to their home base.
- TASK_GOTO_POSITION: The agent must go to a specific location.

A task is associated to its type, the agent that causes the task (the agent itself or the agent which requested it), the position where it should be performed, priority and any possible additional contents. Always the task with the highest priority is launched. Users can redefine the priority of each type of task. Only the agent can add tasks to the list of active tasks, not the user. In Jason, a plan is used to add a task: `add_task`

```
!add_task(task(TaskPriority, TaskType, Agent, Position, Content)),
                                or
        !add_task(task(TaskType, Agent, Position, Content))
```

Such objectives trigger the plan creating the task. The second one assigns the priority defined by the agent.

Execution Loop. Each JGOMAS external agent executes a Finite-State Machine (FSM) as the one in Fig. 3:

- STANDING: The agent does not have any triggered task.
- GO_TO_TARGET: The agent has triggered a task and it is moving to the position where it has to do it.

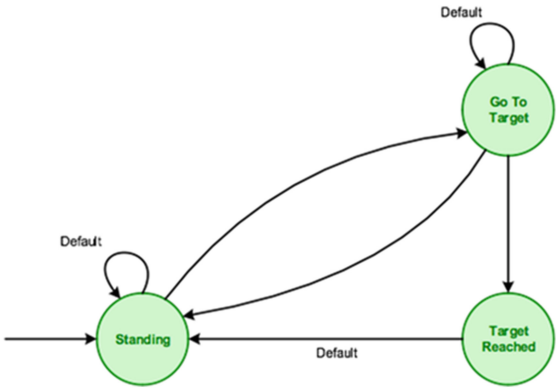

Fig. 3. FSM defining JGOMAS external agents behavior

– TARGET_REACHED: The agent has reached the position where it has to perform the triggered task, and it is performing the actions related to it.

Interface (API). The interface (API) for working with JGOMAS is composed of .asl files written in Jason that contain the different agents (each one with its own beliefs, goals, and behaviors). Inside these files, *jgomas.asl* includes the non-modifiable behaviors of the agent. To modify the behavior of the agent the files with a name following the pattern *jasonAgent_TEAM_TYPE.asl* have to be modified, where TEAM refers to the side of the agent (ALLIED, AXIS), and TYPE is the type of the agent (SOLDIER, MEDIC, FIELDOPS).

Regarding agent beliefs, the main ones are:

– `tasks(task_list)`: Contains the list of active tasks of the agent.
– `fovObjects(object_list)`: Contains the list of objects currently seen by the agent.
– `state(current state)`: This belief is used to indicate the state of the agent in its state machine: *standing*, selecting which task to do or waiting; *go_to_target*, going to its next target; *target_reached*, it has reached the target; quit, has to finish.
– `my_health(X)`: Stores the health of the agent. The initial, and maximum, value is 100. When this value reaches 0, the agent dies.
– `my_ammo(X)`: Stores the amount of bullets of the agent. The initial value is 100.
– `my_position(X,Y,Z)`: Stores the last position known by the agent.

The different plans of the agents that can be modified by the users are:

– `!perform_look_action:` This objective is invoked when the agent looks around and updates the list of surrounding objects *fovObjects(L)*. It would be necessary to implement the plan associated to the creation of this event to be able to look what is around.

- `!perform_aim_action:` This objective is triggered if there is an enemy to aim, which can be used to take a decision about what to do with the aimed agent. A simple implementation of the plan is available.
- `!get_agent_to_aim:` This objective is invoked after *!perform_look_action*, and it would be used to decide if there is any enemy to aim. A simple implementation that can be extended or modified of the associated plan is available.
- `!perform_no_ammo_action:` This objective is triggered when the agent shoots and has no ammo. It is necessary to implement the associated plan to take a decision. For example, to run away.
- `!perform_injury_action` This objective is triggered when the agent is shot. It is necessary to implement the plan associated to the creation of this event to take a decision. For example, run away if the agent has a low life value.
- `!performThresholdAction:` This objective is triggered when the agent has less life or bullets than the thresholds
  my_ammo_threshold(X) and *my_health_threshold(X)*.
  A simple implementation of the associated plan is available, which asks for help to medics or fieldops of its team.
- `!setup_priorities:` This objective is triggered during agent initialization to fix agent task priorities. Each agent has its own priorities. A simple implementation is available. It is interesting to modify it to add new tasks or modify the priorities to have agents that behave in a different way.
- `!update_targets:` This objective can be used to update the tasks and its priorities. It is invoked when the agent changes to the *standing* state and has to choose a new task among the available ones. It is necessary to implement the plan associated to the creation of this event.

Communication. In JGOMAS 2.0, communication is used by agents for registering services and for promoting coordination between them. It is necessary to highlight that an agent plays only one role during the whole game. For example, if an agent A1 is created as a soldier of the type Soldier and belong to the AXIS team, it will be for the rest of the game.

Each role has different features and offers specific basic services that can be improved. Indicating the role and the basic services offered by an agent is carried out at the initialization, using a process known as Registration. Nevertheless, an agent can add new services during the development of the game.

Registration. A role has to register a service to allow other roles to request it. Registration is carried out using the internal action: `.register(‘‘JGOMAS’’, ‘‘type’’)`

Once an agent has registered a service, it is possible for an agent of its same team to check which agents of its team offer a specific service.

Coordination. JGOMAS is provided with mechanisms that allow agent coordination. It can be of one of these two types:

- No communication (implicit): It is achieved by sensing the environment. When an agent looks around itself the objective `!perform_look_action` is

triggered. By rewriting the associated plan it can be decided what to do according to the perception.
- With communication (explicit): In this case it is used the message passing using the following internal action:

  `.send_msg_with_conversation_id (Rec, Perf, Cont, ConvId)`

  where:
  - Rec: receiver of the message (could be a list)
  - Perf: performative (tell, untell, achieve...)
  - Cont: content
  - ConvId: Conversation Id (used in JADE)

3.2 Immersing the Human in JGOMAS

One of the new features of JGOMAS is the interaction with the user. Now, a user can take the control of an agent and give orders directly. The motivations for this interaction are:

- Improve the ability to test the behavior of JGOMAS agents, test strategies, etc. The direct control of an agent allows the user to perform specific actions (like shooting a particular agent or follow a certain path) and see the reaction of the other agents (both enemies and friends) to these decisions. In previous versions, it was necessary that the game reaches this situation by its own evolution to assess its performance.
- From an educational point of view, involvement of the students in the development of agents increases. The students can now validate their designs easier.

This user-JGOMAS interaction is achieved by means of the UNITY engine used in this release. UNITY has allowed not only the improvement in the rendering as discussed above but also to develop the user-JGOMAS interface. Aside from allowing rendering the game, the UNITY render engine gives support to program all aspects of the game including game management, artificial intelligence, etc. In our case, these aspects are developed by JGOMAS code in Java in the JADE architecture or in Jason to implement the agents. However, UNITY can be used for the interface between user and JGOMAS. This way, the user will interact with UNITY which will contact with JGOMAS to communicate user decisions. This communication is done through the JGOMAS messaging system as UNITY is seen as another agent for JGOMAS.

As stated above, the user gives commands via the interface to the agent, but the agent retains its properties. The agent can be any Jason-based agent of the system, and the user does not control it. It sends orders to the agent that are translated to FIPA-ACL messages. These messages are received by the agent and translated into the highest priority intentions, so it may seem that the user controls it, but if the user stops to send orders to the agent, it will still work.

The game may have multiple users each one controlling an agent. One consequence of this is an increasing immersion of the user. The user does not know if the agents he is fighting against (or collaborating with) are under the control of

Fig. 4. Human immersed in one JGOMAS soldier

another user or not. Also it can be considered double immersion because it also acts on the system side: the agents do not know which agents are under computer control or under the control of external users. The agents act according to their beliefs and perceptions and the subsequent reasoning based on them, not considering the nature of the other agents. To make this interaction, the user selects an agent by means of the mouse. The user can take the control of this agent by pressing the key B. From that moment the user can control the agent through a series of keyboard commands and the view is centered in this agent's viewpoint (see Fig. 4). Table 1 summarizes user commands. The choice of the keys did not follow naming criteria but the keyboard layout.

One of the most interesting options with respect to validation is the action associated to the P key. This key sends a message from the user-controlled agent to another agent. The user only has to implement the Jason code as the receiving agent wants to respond to that message. Another option to note is the activation of the first-person camera that further enhances the user's immersion in the game.

Table 1. User commands

Key	Action
B	Control agent selected
I	Move agent in the current direction
J	Turn the agent to the left
L	Turn the agent to the right
K	Turn the agent 180 degrees
V	Shot
C	Call for medical
X	Call for ammo
Z	Call for backup
T	Select the upper camera
M	Toggle to the first person camera
N	Toggle to the third person camera
G	Show controlled agent info
P	Send a message to another agent

4 Conclusions and Future Work

This paper has presented JGOMAS 2.0, which is a simulation game where students design and implement a group of agents and test different coordination strategies in order to win the game. This new version improves different aspects regarding the first version. Concretely includes the possibility to develop Jason agents allowing students to integrate BDI-oriented strategies in the decision-making of the agents. Moreover, the new version has introduced the human in the execution process allowing the student to interact with his/her developed team. This interaction includes sending messages or participating as another member of the team. As future work, emotional states will be included in the agent's model. These emotional states will change according to changes in the environment and the agent's personality. Emotional values may be used by the students in order to improve the decision making of the agent's team during the game.

Acknowledgments. Work partially supported by Spanish Government through the project iHAS (grant TIN2012-36586C03-01).

References

1. Overmars, M.: Teaching computer science through game design. Computer **37**(4), 81–83 (2004)
2. Barella, A., Valero, S., Carrascosa, C.: JGOMAS: New approach to AI teaching. IEEE Trans. Educ. **52**(2), 228–235 (2009)

3. Bellifemine, F.L., Caire, G., Greenwood, D.: Developing Multi-Agent Systems with JADE. Wiley Series in Agent Technology. John Wiley & Sons, New York (2007)
4. Bordini, R.H., Hubner, J.F., Wooldridge, M.J.: Programming Multi-Agent Systems in AgentSpeak using Jason. J. Wiley, Chichester, Hoboken (2007)
5. Rao, A.S.: Agentspeak(l): Bdi agents speak out in a logical computable language. In: Perram, J., Van de Velde, W. (eds.) MAAMAW 1996. LNCS, vol. 1038, pp. 42–55. Springer, Heidelberg (1996)
6. Bellifemine, F., Poggi, A., Rimassa, G.: Developing multi-agent systems with JADE. In: Castelfranchi, C., Lespérance, Y. (eds.) ATAL 2000. LNCS, vol. 1986, pp. 89–103. Springer, Berlin Heidelberg (2001)
7. O'Brien, P.D., Nicol, R.C.: Fipa towards a standard for software agents. BT Technol. J. **16**(3), 51–59 (1998)