Situational Programming: Agent Behavior Visual Programming for MABS Novices

Fabien Michel[1], Jacques Ferber[1], Pierre-Alain Laur[2], and Florian Aleman[2]

[1] LIRMM Lab. d'Informatique, Robotique et Micro-électronique de Montpellier
CNRS - Université Montpellier II, 161 rue Ada 34392 Montpellier Cedex 5 - France
{fmichel,ferber}@lirmm.fr
[2] FEERIK, Inc. 91 rue Font Caude 34080 Montpellier - France
{pal,florian}@feerik.com

Abstract. This paper presents an agent-oriented visual programming approach which aims at providing MABS end-users with a means to easily elaborate artificial autonomous behaviors according to a targeted domain, namely *situational programming* (SP). More specifically, SP defines design principles which could be used to develop MABS visual programming toolkits suited for non developers and MABS novices. This paper presents SP and how it is used to build a MABS video game which can be played by MABS novices, that is any Internet user.

Keywords: MABS, Agent-Oriented Programming, Visual Programming, Situational Programming, Video Game.

1 Introduction

Multi-Agent Based Simulation (MABS) is used in various research and application domains such as social science, ecology, ethology, etc. Considering this interdisciplinary aspect, one crucial issue is that many MABS end-users are not professional programmers while most MABS toolkits require computer programming skills [1].

Many MABS toolkits tackle this issue by defining coding primitives that help to design autonomous behaviors with respect to a targeted domain (e.g. Cormas [2]). Still, even if such approaches may hide to some extent the complexity of high level programming concepts such as inheritance, one has still to be familiar with basic programming concepts such as conditional expressions (e.g. if-then-else statement), looping structures (e.g. for, while, etc.) and variable affectations. Additionally, a textual programming syntax remains to be learned in every case. Therefore, efforts have been done to provide MABS toolkits with visual programming (VP) features so that they require few or no programming knowledge. Indeed, even if VP is naturally not as flexible as textual programming, it represents a very interesting solution considering the use of MABS by non professional programmers.

This paper introduces and discusses a VP variant for designing artificial behaviors, namely *Situational Programming* (SP). The goal of SP is to focus on allowing artificial behavior programming without any programming skill nor MABS

T. Bosse, A. Geller, and C.M. Jonker (Eds.): MABS 2010, LNAI 6532, pp. 1–15, 2011.

modeling ability. To this end, SP does not intend to allow novice developers to build a MABS from scratch. SP rather defines design principles which could be used by programmers to develop MABS VP toolkits and graphical user interfaces (GUI) fulfilling this requirement. The outline of the paper are as follows. The next section discusses the motivations and advantages of VP approaches for MABS, presents some existing tools and then highlights the limitations of existing approaches. Section 3 presents the motivations and underlying ideas of SP. Section 4 details a SP case study which has been done in the scope of a MABS video game. Section 5 highlights the limitations of the proposed approach. Section 6 discusses some related research works and Sect. 7 concludes the paper.

2 Visual Behavior Programming for MABS

2.1 Motivations

VP enables MABS end-users to create behaviors by manipulating graphical programming elements which eventually represent textual programming blocks. In most cases, the graphical elements are connected by arrows representing relations such as actions ordering, condition statements, loops and so on. So, users have to build a diagram that will represent the agent behavior and do not have to know about the programming language which is used under the hood.

Another major interest of VP is to disentangle the user from the language syntax complexity and requirements. Indeed, users cannot do programming syntax errors since graphical elements represent only valid programming statements. Moreover, VP tools further help users as they usually embed a specific spatial grammar that does not permit invalid connections between graphical elements nor invalid states for the components.

The main advantages of VP tools are therefore twofold:

1. Knowing about the programming language which is used in the platform is not a requirement: Novice developers can thus use the simulation tool.
2. Syntax correctness could be ensured thanks to the grammar which could be embedded in the graphical elements and in the manner they could be defined, combined or connected.

2.2 Examples of Visual Programming for MABS

This section presents three examples of MABS platforms which have VP features: (1) AgentSheets, (2) SeSAm and (3) Repast Simphony.

AgentSheets Developed in the early nineties, the seminal idea of the end-user programming tool AgentSheets relied on building a new kind of computational media allowing casual computer users to design complex interactive simulation [3]. So, the philosophy of this environment is to hide as much as possible the complexity of simulation-authoring mechanisms, thus focusing on the idea that simulation could be fruitfully used as an interesting cognitive thinking tool in many domains. Especially, AgentSheets is mainly used for educational purposes.

The VP language of AgentSheets is called Visual AgenTalk (VAT). VAT is a rule-based language allowing users to express agent behavior as if-then rules containing conditions and actions. Additionally, VAT enables what is called *Tactile Programming* and adds interactivity on the program representation: Program fragments can be manipulated through drag and drop operations to compose behaviors and also help end-users to explore and test agent behaviors thanks to several operations obtained on mouse clicks. AgentSheets is today an ongoing commercial tool[1] which has also been extended to a 3D version named AgentCubes [4].

SeSAm (Shell for Simulated Multi-Agent Systems) is a Java open source MABS tool which aims at providing a generic environment for modeling and simulating MABS [5]. SeSAm[2] embeds several VP tools that help for different tasks considering the modeling of agent behavioral processes.

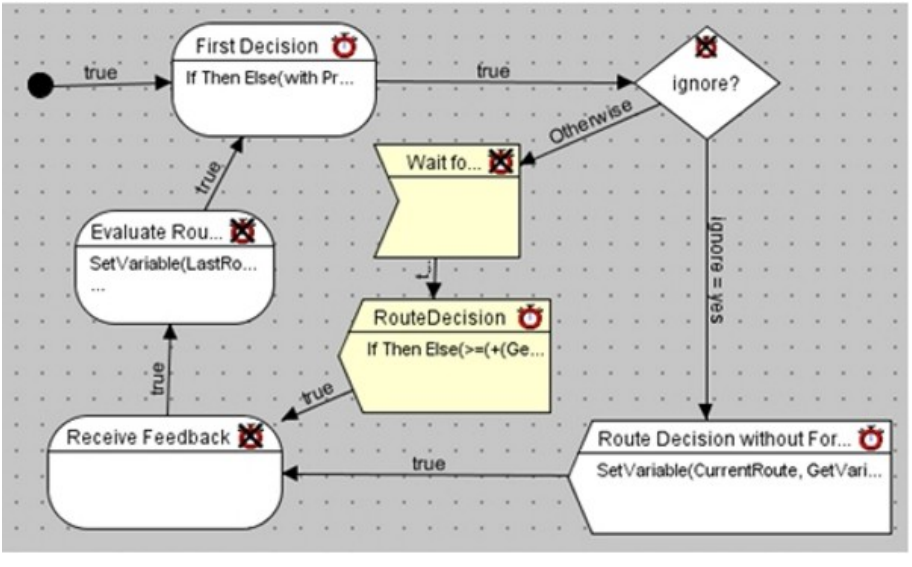

Fig. 1. Screenshot from the SeSAm implementation of a driver agent in [6]

For instance, interactive dialog elements enable to specify (potentially nested) primitive calls which could be used for specifying behavioral rules, creating initialization procedures, and so on. These interactive dialogs not only hide the use of Java but also dynamically check the underlying validity of the primitives (e.g. type-consistency). So, behaviors can be completely programmed without knowing the subtleties of an object oriented language such as Java. In SeSAm,

[1] http://www.agentsheets.com
[2] http://www.simsesam.de

VP is also used to visualize and specify the behavior of an agent using an activity diagram (Fig. 1): Activity nodes are linked with arrows describing transition rules between activities.

Repast Simphony (RS) is an open source generic MABS toolkit which is used in various application domains. RS extends the basic Repast[3] platform to provide advanced visual modeling and programming features for novice developers [7]. From a technical point of view, RS is a preconfigured Eclipse-based IDE (Integrated Development Environment) that notably uses the Flow4J[4] Eclipse plugin which enables to model process flows in a drag and drop manner.

As Fig. 2 shows, the Repast agent editor could be used to create a diagram using various block types (task, perceptions, condition evaluation, loop, etc.) which are connected by links defining the agent behavior logic.

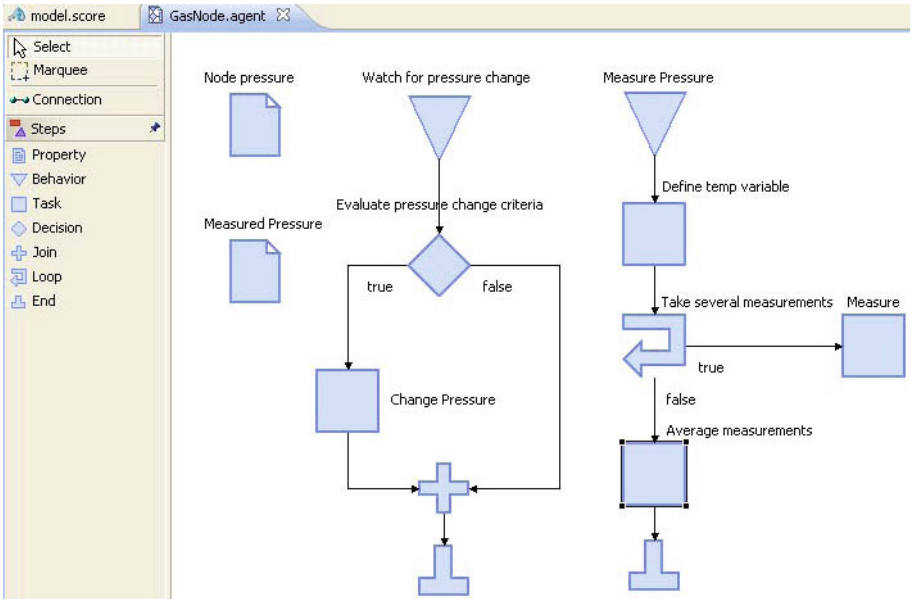

Fig. 2. The visual agent behavior editor of Repast Simphony

Thanks to this approach, the user does not need to write any line of code: When saved, the diagram is automatically translated in a traditional programming language and the behavior could be directly used within the simulation.

2.3 Limitations of Existing Approaches

Behavior Complexity. As remarked in [8], using existing MABS visual tools, building complex agents' cognitive models is clearly a tricky and painful task

[3] http://repast.sourceforge.net
[4] http://flow4jeclipse.sourceforge.net

because they usually allow to manipulate only basic concepts. To overcome this issue within the scope of social science, the authors have proposed a visual modeling language relying on the INGENIAS methodology [1]. This work allows to visually model complex behaviors based on intentional and organizational concepts. Still, the proposed modeling language is rather sophisticated as it intentionally targets MABS modeling experts, especially social scientists. Nonetheless, this work clearly highlights the complexity issue of existing MABS VP tools.

Behavior Graphical Representation. The difficulty of visually programming complex behaviors does not only come from the relative simplicity of the concepts usually used. Indeed, a critical issue which comes with existing MABS VP tools is to graphically represent complex behaviors. Obviously, a diagram containing more than about twenty graphical elements could not be really explicit nor intuitive and thus hardly understandable at first sight. Moreover, even if one can (1) reduce the size of each graphical element using explicit iconic symbols or (2) use nested components, the screen size will still be a limitation in itself. Therefore, it is crucial to provide end-users with behavioral graphical abstractions which enable simple and synthetic presentations of the behavior logic.

A Programmer's Mindset is Still Needed. Here, we want to emphasize on another issue which has been found crucial from our point of view: Even when a very high level of abstraction is considered, MABS VP tools still involve basic programming concepts. Especially, despite the intuitive aspects of MABS VP tools, the end-user has still to deal with concepts such as if-then-else statements, loops, variables usage and so on. In an educational context, this could be not a problem since the goal may be precisely to teach computer programming, or even agent-based programming concepts, to students. However, this remains a serious issue considering MABS end-users who are novice developers: Ultimately, they need to know and understand some fundamental computer programming concepts.

Therefore, we think that there is room for MABS VP tools that do not use any traditional programming concepts nor complex modeling features. The next section presents an approach which has such a goal, namely *Situational Programming* (SP).

3 Situational Programming

3.1 Objectives and Requirements

The main goal of SP is to provide non developers and MABS novices with a means to easily elaborate and test artificial behaviors with respect to a targeted domain. Obviously, such a goal requires to consider a VP approach. Additionally, we want our approach to overcome as much as possible the limitations discussed previously. Therefore, SP should also stick to the following requirements:

– End-users should not face any traditional programming concepts nor complex modeling tasks.
– The behavior logic representation should not take too much space on the screen and thus be as synthetic as possible.
– It should be still possible to define complex behavior logics.

So, even if we consider a VP-based approach, according to these requirements our focus is so that, contrary to the approaches we discussed, our current goal is not to provide a tool allowing non developers to build a MABS from scratch. The purpose of SP is rather to define agent-oriented programming principles which could be used to develop MABS VP tools which concretelly fulfill these requirements according to specific targeted domains.

3.2 Principle of the Approach

As we will now explain, inspirations for SP rely on observations and remarks about agent behavior programming in general, not only with respect to behavior VP. So, let us first consider a synthetic and traditional model of an agent behavioral process illustrated in Fig. 3: (1) perception, (2) deliberation, and then (3) action [9]. From a technical point of view, programming an agent behavior relies on (1) parsing perceptions, (2) using the obtained results in the deliberation, and (3) then take a particular decision based on deliberation, that is an action on the environment.

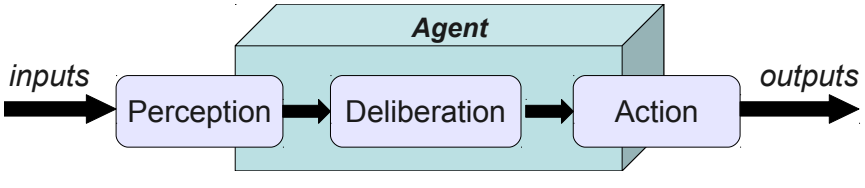

Fig. 3. The behavior of an agent: (1) Perception (2) Deliberation (3) Action

One has to remark that, considering a set of raw and basic percepts, it could be very difficult to programmatically define a relevant view of the world state. In other words, one has to first work on the percepts just to build structured data on which the deliberation could reason on. So, the programming complexity of this task rapidly grows according to the agent cognition level required. For instance, it is very difficult for a soccer robot to catch all the aspects of the current situation starting from its inputs [10].

Similarly, the same observation also holds about complex actions or plans: They are tricky to program starting from a set of basic actions manipulating only the effectors of the agent. So, in order to fulfill its objectives, SP is based on an approach wherein the perception and action phases are considered as too complex processes for MABS novices. Consequently, SP relies on preworking these phases so that end-users can focus on the deliberation part of the agent.

3.3 Very High Level Perceptions and Actions

The complexity of using raw percepts, and thus building relevant perceptions on which the agent could reason on, is very interestingly discussed in [11]. This work proposes a cognitive architecture composed by three layers: (1) *Reality*, (2) *Concept* and (3) *Mind*. The concept model layer is particularly of interest as it is in charge of mapping the physical environment reality to high level concepts which could be easily and efficiently used by the agent mind layer to deliberate. In other words, the goal of the concept layer is to allow agent minds to understand reality. The main idea underlying SP is related to such a conceptual philosophy.

The idea is to provide end-users with very high level percepts defining domain-oriented situations. By *situation*, we mean a combination of the possible states of high level percepts on which one has only to deliberate to choose an action: All the percepts compilation work is already done. For instance a percept could be *being under attack* or *dribbling the ball*, and the state within the situation *true, false*, or *ignored*. The end-user has thus only to select the state of each percept for defining situations.

The same philosophy is also used to define the actions that end-users will be able to select according to a particular situation. So, instead of basic actions, very high level plans are defined using a combination of easily tunable domain-oriented actions which are predefined. For instance, a plan could be *patrolling-an-area* (main plan) using a *sinusoidal move* (plan parameter). Figure 4 gives an abstract sketch of this approach.

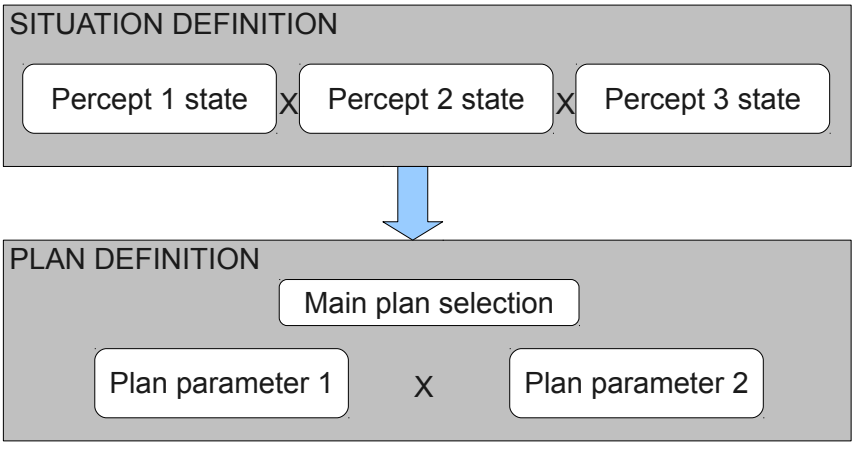

Fig. 4. Situational Programming Sketch

Of course, with respect to this approach, end-users should be able to define as many situations as they want. However, SP relies on the idea that it is very important for end-users to focus on defining *only one* situation/plan couple at a time. Following such a requirement disentangles the action selection part from

the situation classification part. As we will see, this seriously reduces the complexity of both the end-user's work and the behavior graphical representation. Therefore, it is crucial to build the GUI with this requirement in mind. We will discuss how conflicts between situations are resolved by the user later on.

The basic work of the end-user is thus easy and straightforward: (1) tune the proposed percepts to define a particular situation and (2) then decide of a corresponding parameterizable plan. So, by defining several situation/plan couple matching, the end-user will define, and thus program, the agent behavior by reflecting his point of view and thus literally materializing his way of thinking.

3.4 Achieving Complex Behaviors without GUI Complexity

At first glance, the number of possible situation/plan combinations is the only factor of complexity. So, SP may look quite limited in terms of behavioral complexity since situations and plans are predefined. However, SP also includes additional design principles that allows behavioral complexity at a very low cost in terms of graphical user interface (GUI) complexity and understandability.

Situation Definition. As previously mentioned, the state of each percept contributes in defining a situation. The simplest percepts are related to the veracity of a particular fact and only have three states: (1) true, (2) false or (3) ignored (e.g. being under attack). State selection is done by single clicks that cycle through these three states.

Considering percepts related to a quantitative value such as *energy*, the state of the percept (e.g. *low energy* or *high energy*) must be defined according to a threshold (e.g. 50%). Such a threshold could be internally defined and not visible to the user. But, when explicitly presented, we give the user the opportunity of manually choosing this threshold using a usual graphical slider. We identified such percepts as *thresholded* and all others as *boolean*. So, when a thresholded percept is involved in a situation, this virtually increases the number of potential situations to infinity, thus introducing more behavior complexity without increasing GUI complexity. Also very important is the fact that this introduces singularity and heterogeneity among the programmed behaviors.

Plan Parameterization. Plans are defined from a very high level domain-oriented perspective. For instance, the plan *patrolling-an-area* may have two parameters: (1) the location and (2) the type of move (sinusoidal, straight lines, etc.). Much complexity could result from how plans are parameterizable. Still, this aspect is fully domain-oriented and not generalizable. However, we found an interesting parameter which could be applied in any domain. Indeed, one problem we found in the early stages of this research was that, in some cases, agents were constantly changing of selected plan: One situation may disappear and reappear in a very short period of time. To overcome this problem, we introduce an additional generic parameter to plans: *Stubbornness*. Stubbornness defines how much time the agent should stick to the selected plan without reconsidering situation. So, the stubbornness parameter solves the well know problem

of persistence/commitment in action selection [12]. Stubbornness also introduces another level of complexity for the behaviors and increases their singularity, giving agents personality.

Situation/Plan Couples Priorities. An important aspect of SP which has not been discussed so far is that of situational conflicts. Indeed, it turns out that according to the way situations are defined by the user, several situations could simultaneously match the actual world state. Therefore it is important to provide end-users with a means of prioritizing situations against each other through a simple graphical presentation. In this respect, a column in which situations are ordered was found as the most appropriate solution. Moreover, such a presentation allows the user to prioritize the situations using a drag and drop approach. Once again, this introduces another behavioral complexity level without increasing the complexity of the GUI.

4 Applying SP: The Warbot Video Game

4.1 History and Objectives

Warbot[5] is a MABS game wherein teams of autonomous bots fight against each other. Historically, Warbot was designed ten years ago using the MadKit platform [13] to teach high school students agent-oriented programming through a competition: Students have to program the minds of Warbot agents whose bodies are predefined so that the best behaviors win the battle.

In collaboration with the Feerik[6] company, which is specialized in free-to-play on-line video games, we are developing a Warbot version that could be played by anyone, especially non programmers according to the Feerik's business model: A Feerik end-user could be any Internet user.

4.2 Warbot Domain-Oriented Percepts and Plans

As discussed in Sect. 3, to provide end-users with a SP-based MABS toolkit, developers have to first identify the related domain-oriented percepts and plans that will be used to program the behaviors.

Based on the experiences we had with the student version of Warbot, we identified a preliminary set of 5 high level percepts which were regularly programmed throughout the passed years:

- *Energy level.*
- *Number of detected enemies.*
- *Being under attack.*
- *Being helped by a teammate.*
- *Being asked for help by a teammate.*

[5] www.madkit.net/warbot
[6] www.feerik.com

Similarly, we identified 4 high level plans:

- *Go to point*
- *Patrol zone.*
- *Fight.*
- *Flee.*

Here, one may wonder why there is not a *help teammate* plan. In fact, it has been decided that this would be a parameter of all plans. That is, doing something, a bot can decide to take into account or not its teammates.

4.3 Warbot GUI

Figure 5 presents portions of the actual version of the Warbot behavior editor. The zone 1 presents the percepts which can be selected by the user to define a situation by dragging and dropping them in the zone 2. The zone 2 also permits to give a name to the current situation. In the third zone, the user can parametrize the plan which is associated with the current situation. The fourth zone summarizes the situations which have been already defined and also enables the user to prioritize them using drag and drop moves.

Figure 5 also shows that, defining situations in Warbot, it is possible to select different states of the same percept to associate a particular plan to several situations at once. In this example, the bot will fight whatever the number of enemies if it has more than 50% of energy.

The Warbot behavior programming GUI intentionally does not yet include all the Warbot domain-oriented percepts and plans discussed earlier. Indeed, it is planned to incrementally introduce each percept in the game in order to teach the end-users how to program complex behavior step by step: Simplicity at first glance is a major requirement for Feerik. Moreover, acquiring progressively new features is part of Feerik's business model.

Technically, the Warbot behavior GUI is made in Macromedia Flash. End-user's inputs are then compiled and used by the TurtleKit MABS platform [14] which in turn produces a game instance rendered by the Unity 3D web player in the user's web browser as shown in Fig. 6. Other web pages are used to define which bots compose the team and what are their equipment in terms of legs, arms, head, and weapon.

4.4 First Feedbacks and Remaining Work

Although game designers have not worked on the GUI design, first feedbacks from novice developers are very encouraging since they do easily find their way in developing artificial bot behaviors. Especially, they do appreciate that (1) situations can be defined by single mouse clicks and that (2) plans can be selected and parameterized using simple forms. Also, they find quite intuitive the use of a drag and drop approach to define priorities between situations.

The game-play is a major aspect of video game. So, one important work which remains to be done is to give the end-user statistical feedbacks about how his

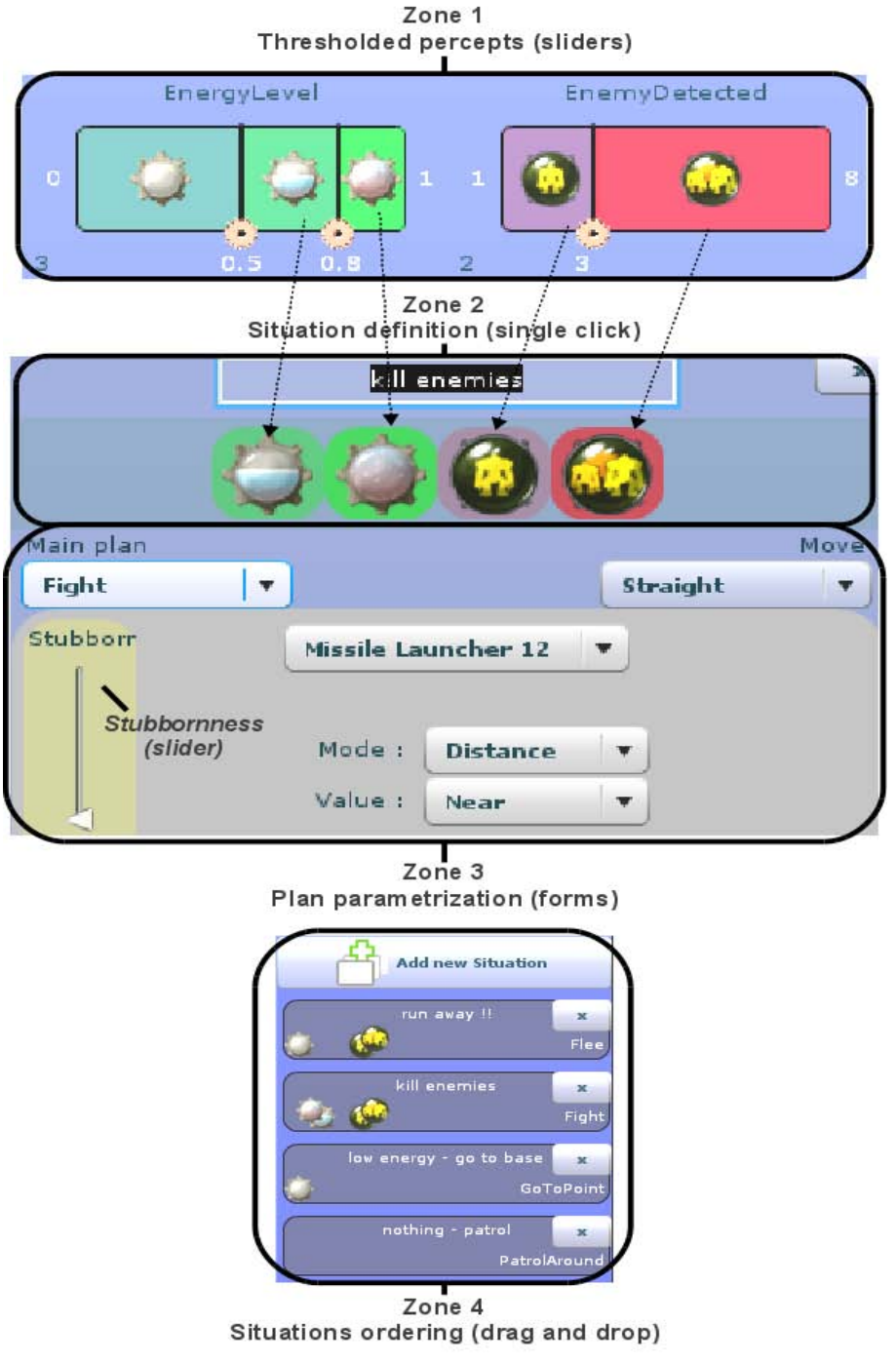

Fig. 5. Situational Programming with the Warbot GUI

Fig. 6. The Warbot video game rendered by the Unity 3D web player

bots behaved during a simulation, in order to provide the end-user with a means to identify the strengths and weaknesses of the defined behaviors. To this end, it is planned that the game will collect information such as which plans have been used, how much time, how much energy has been consumed during each plan, and so on.

5 Current Limitations of the Approach

As previously stated, SP does not intend to allow novice developers to build a MABS from scratch. Therefore, considering each targeted domain, SP of course requires that a true developer has programmed priorly all the different aspects of the corresponding SP-based MABS toolkit. Especially, this requires that the model be developed in close collaboration between the developers, on one hand, and researchers as well as householders, who know the real system, on the other hand. Still, one has to admit that this is true to some extent for any MABS platform. Moreover, we are actually tackling this issue at a software engineering level so that the underlying simulator software structure could be easily reused for new application domains.

Outside the scope of our current objectives, another limitation relies on the fact that end-users cannot add new percepts nor modify how they are defined. The same remark holds about plan structure. Still, we think that this is not a definitive limitation, at least considering percepts building. For instance, end-users may have access to another editor page which would consist in representing the characteristics of the environment. In such a page, the end-user could select some properties to define new usable percepts in just a limited number of clicks.

6 Related Works

Globally considering the issue of programming complex but efficient artificial behaviors, an interesting trend of research consists in tracking and learning how humans play a game in order to imitate their behaviors programmatically. For instance, in [10], considering the programming of Robosoccer agents, the authors track how a human controls a Robosoccer agent in order to model his behavior using machine learning techniques. SP could also be viewed as a means to take advantage of the human mind to build complex artificial behaviors. In this respect, a very interesting aspect of SP is that the reasoning of the human user has not to be programmed: It is entirely embedded in the resulting behaviors. For instance, with many players using the Warbot game, it will be possible to extract recurrent programming patterns which will be very interesting to study.

Obviously, there is an apparent conceptual link between participatory design of MABS (e.g. like in [15]) and SP since the behavior of the artificial agent partly remains inside the end-user's mind. So, both approaches strongly involve the end-user in the simulation design process as they both try to somehow capture and then compute his way of thinking. Still, these two are quite different, in terms of both objective and design work flow. Indeed, in a participatory mode, the user plays a role during the simulation, which is for instance incompatible with the game-play's objectives of Warbot. Moreover, while role-playing, the user does not program anything. In SP end-users have to program off-line all the behavior of the agent and thus cannot control its behavior while the simulation is in progress. Nonetheless, it is clear that SP and participatory design of MABS certainly share some common concerns which could be fruitfully exhibited.

The idea of using very high level concepts for simulation programming purposes is not new. For instance, in [16], the authors investigated the suitability of a very high level language (SETL [17]) for simulation. They said that such a language is one which incorporates complex structured data objects and global operations upon them. Obviously, SP relies on a similar philosophy. Therefore, it would be interesting to study how the design principles related with very high level language could be translated in our approach.

Finally, with respect to the idea of simplifying MABS programming, it would be interesting to study how SP could be coupled with the approach proposed in [18]. This approach, called *Interaction-Oriented Design of Agent simulations* (IODA), relies on defining a MABS as an interaction matrix specifying how the interactions between the agents take place. So, once a set of possible interactions is specified and programmed, defining the behavior of the agents simply consists in selecting, through a GUI representing the matrix, the interactions in which the agents could be involved. So, thanks to a visual interface that does not involve any programming concepts, this approach allows MABS novices to design and test numerous simulation models without any programming effort as end-users.

7 Conclusion

This paper has presented an agent-oriented programming approach which aims at providing MABS end-users with a means to easily elaborate artificial autonomous behaviors according to a targeted domain, namely *Situational Programming*. More specifically, SP defines design principles which could be used to develop MABS VP toolkits suited for non developers and MABS novices. So, following a MABS VP approach, one of the main interests of using a SP-based approach is to allow behavioral complexity with GUI simplicity.

We showed how SP is applied in the scope of a SP-based MABS online video game and, even if some work remains to be done on the Warbot GUI aesthetic, the first feedbacks we had are promising and showed us that SP is a concrete solution allowing any computer user to program artificial behaviors.

Among the related research perspectives we discussed in the previous section, collecting player data to study behavior programming patterns is of particular interest. Firstly, this will enable us to increase the game play of the Warbot game by rewarding the best players. Secondly, from a more general perspective, we think that SP could be a concrete alternative to participatory design or human imitation approaches for MABS. SP can be used as a means to *translate* human behaviors into computer programs. This in order to (1) study their characteristics in silico or (2) integrate realistic behaviors into MABS experiments.

References

1. Pavón, J., Sansores, C., Gómez-Sanz, J.J.: Modelling and simulation of social systems with INGENIAS. Int. J. Agent-Oriented Softw. Eng. 2(2), 196–221 (2008)
2. Bousquet, F., Bakam, I., Proton, H., Page, C.L.: Cormas: Common-pool resources and multi-agent systems. In: Proceedings of the 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, pp. 826–837. Springer, Heidelberg (1998)
3. Repenning, A., Ioannidou, A., Zola, J.: AgentSheets: End-user programmable simulations. Journal of Artificial Societies and Social Simulation 3(3) (2000)
4. Ioannidou, A., Repenning, A., Webb, D.C.: AgentCubes: Incremental 3d end-user development. Journal of Visual Language & Computing 20(4), 236–251 (2009)
5. Klügl, F., Herrler, R., Fehler, M.: SeSAm: implementation of agent-based simulation using visual programming. In: AAMAS 2006: Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems, pp. 1439–1440. ACM, New York (2006)
6. Klügl, F., Bazzan, A.L.C.: Route decision behaviour in a commuting scenario: Simple heuristics adaptation and effect of traffic forecast. JASSS, The Journal of Artificial Societies and Social Simulation 7(1) (2004)
7. North, M., Tatara, E., Collier, N., Ozik, J.: Visual agent-based model development with Repast Simphony. In: Agent 2007 Conference on Complex Interaction and Social Emergence, Argonne, IL, USA, Argonne National Laboratory, pp. 173–192 (November 2007)
8. Sansores, C., Pavón, J., Gómez-Sanz, J.J.: Visual modeling for complex agent-based simulation systems. In: Sichman, J.S., Antunes, L. (eds.) MABS 2005. LNCS (LNAI), vol. 3891, pp. 174–189. Springer, Heidelberg (2006)

9. Michel, F., Ferber, J., Drogoul, A.: Multi-Agent Systems and Simulation: a Survey From the Agents Community's Perspective. In: Weyns, D., Uhrmacher, A. (eds.) Multi-Agent Systems: Simulation and Applications. Computational Analysis, Synthesis, and Design of Dynamic Systems, pp. 3–52. CRC Press - Taylor & Francis (June 2009)
10. Aler, R., Valls, J.M., Camacho, D., Lopez, A.: Programming Robosoccer agents by modeling human behavior. Expert Systems with Applications 36(2, Part 1), 1850–1859 (2009)
11. Chang, P.H.M., Chen, K.T., Chien, Y.H., Kao, E.C.C., Soo, V.W.: From reality to mind: A cognitive middle layer of environment concepts for believable agents. In: Weyns, D., Van Dyke Parunak, H., Michel, F. (eds.) E4MAS 2004. LNCS (LNAI), vol. 3374, pp. 57–73. Springer, Heidelberg (2005)
12. Tyrrell, T.: The use of hierarchies for action selection. Adaptive Behavior 1(4), 387–420 (1993)
13. Ferber, J., Gutknecht, O., Michel, F.: From agents to organizations: an organizational view of multi-agent systems. In: Giorgini, P., Müller, J.P., Odell, J.J. (eds.) AOSE 2003. LNCS, vol. 2935, pp. 214–230. Springer, Heidelberg (2004)
14. Michel, F., Beurier, G., Ferber, J.: The TurtleKit simulation platform: Application to complex systems. In: Akono, A., Tonyé, E., Dipanda, A., Yétongnon, K. (eds.) Workshops Sessions, First International Conference on Signal & Image Technology and Internet-Based Systems SITIS 2005, pp. 122–128. IEEE, Los Alamitos (2005)
15. Guyot, P., Honiden, S.: Agent-based participatory simulations: Merging multi-agent systems and role-playing games. Journal of Artificial Societies and Social Simulation 9(4), 8 (2006)
16. Franta, W.R., Maly, K.: The suitability of a very high level language (setl) for simulation structuring and control. In: Robinet, B. (ed.) Programming Symposium. LNCS, vol. 19, pp. 156–169. Springer, Heidelberg (1974)
17. Schwartz, J.: Set Theory as a Language for Program Specification and Programming. Courant Institute of Mathematical Sciences, New York University (1970)
18. Kubera, Y., Mathieu, P., Picault, S.: Interaction-oriented agent simulations: From theory to implementation. In: Ghallab, M., Spyropoulos, C., Fakotakis, N., Avouris, N. (eds.) Proceedings of the 18th European Conference on Artificial Intelligence (ECAI 2008), pp. 383–387. IOS Press, Amsterdam (2008)