Prolog-Based Reasoning Layer for Counter-Strike Agents

Grzegorz Jaśkiewicz

Warsaw University of Technology
The Faculty of Electronics and Information Technology,
ul. Nowowiejska 15/19 00-665 Warsaw Poland
grzegorz@jaskiewi.cz

Abstract. In this paper it is shown an application of an agent-oriented programming paradigm with a reasoning layer based on a logic programming. The presented solution was used to build bots in *Counter-Strike* video game. The software architecture for new bots has been presented and benefits were discussed. This article does not discusses all the work in detail, but provides more general idea about research, which was done.

Keywords: Agent systems, Prolog, Reasoning, Artificial Intelligence, Bots, Video Games

1 Introduction

Counter-Strike (CS) is *First Person Shooter* game (FPS). Characteristic of this game genere is observation of a game world from first person perspective (see. 1) [?].

Fig. 1. The first person perspective in *Counter-Strike*.

Counter-Strike is a very popular video game, which is meant to be played exclusively in a multiplayer mode. In each game, players are divided into 2 teams

- terrorists (TT) and counter-terrorist (CT). Both teams fight against each other using modern firearms. Each game lasts 30 minutes by default and consists of several 5 minute rounds. There are several map types which affect goals for both teams.

AS assasination. One of CT's becomes VIP, which must reach the safe zone. TT's try to eliminate the VIP.
CS hostage rescue. CT's try to move hostages to rescue zone.
DE bomb defusion. TT's try to plant a bomb, CT's may try to defuse it afterwards.
ES escape. TT's try to flee to the escape zone, CT's hunt them.

If team is successful in achieving its goal, then the team wins the round. Elimination of the entire opposing team usually also means a victory.

The gameplay of Coutner-Strike is very realistic, eliminated players do not respawn immediately as in most FPS multiplayer games and each player can be eliminated with only few well-aimed bullets. Therefore, Counter-Strike is a very team-oriented game, where cooperation is more important than skills of individuals players. Players do often form groups called *clans*, who play together against other clans. There are even organized tournaments in Counter-Strike. The game has rank of official sport in some countries [?].

Unfortunately, the game in its early versions was completely unplayable for a single player, because it was meant to be played entirely in multiplayer mode. Community of the game fans developed game extensions, which contained computer-controlled players - bots [?]. CS Bots creators had difficult problem to solve, because developed bots had to act as team and co-operate.

In this article it has been show extension of existing bot, using foundations of multiagent systems. Note that, concept of application agent-oriented programming in FPS video games is not a new one e.g [?], [?], [?]. However, there are different implementations of this concept leading to different bot architectures. One of the approaches to bot programming based on this concept is providing scripting language e.g [?], [?], so that it is easy to experiment with various behaviors, bot interactions and tactics. This approach relies on expert knowledge, which is merged into script or program source code. Completely different approach is to involve machine learning and let bots gain knowledge through interaction with other players. There is already some research in this area [?], [?], [?]. Due to complex nature of the problem ML techniques do not solve it entirely, but serve as support for techniques involving domain knowledge about game. In this work expert knowledge based approach is applied. One of the knowledge representations could be first order logic. This representation is declarative, thus readable to human. This was main motivation for choosing Prolog as scripting language. Presented AI bots component is divided into main 2 layers:

– reasoning layer - high-level decision making, corresponds to human declarative knowledge,
– execution layer - low-level decision making, corresponds to human procedural knowledge.

A similar approach could be found in autonomous robot programming, e.g in *Robocup* competition [?] [?].

2 Material and Methods

2.1 Formal description

Let consider game environment state g at some moment fixed in time t as set facts. Let $\mathbb{F}$ denote whole universe of possible facts. So that $g(t) \in P(\mathbb{F})$. Each agent $b_i \in \mathbb{B}$ has knowledge at any point of time about some subset of those facts. Define

$$K : \mathbb{B} \times P(\mathbb{F}) \rightarrow P(\mathbb{F})$$

as knowledge operator. In this model internal states of bot agents could be fact p_i which satisfy following necessary condition

$$\exists! b_i \in \mathbb{B} \quad p_i \in K(b_i, s_t)$$

fact p_i known to all bots satisfy following condition

$$\forall b_i \in \mathbb{B} \quad p_i \in K(b_i, s_t)$$

Time passed from beginning of the round could be example of such fact. At each timepoint any of n bots b_i can make decision $d_i \in D$ and next game state is dependent of all decisions made by bots. This transition is handeled by game enviornment, which could be described as a function G_e.

$$G_e : P(\mathbb{F}) \times \underbrace{D \times D \times \ldots \times D}_{n} \rightarrow P(\mathbb{F})$$

Therefore low-level action is function

$$A_l : \mathbb{B} \times P(\mathbb{F}) \rightarrow D \times \underbrace{\{\text{success}, \text{fail}, \text{continue}\}}_{\text{action status}}$$

and high-level reasoning is a function

$$A_h : \mathbb{B} \times P(\mathbb{F}) \rightarrow \{ \underbrace{A_l}_{\substack{\text{current} \\ \text{low level action}}} \times \underbrace{\mathbb{F}}_{\substack{\text{motivation} \\ \text{for action}}} \times \underbrace{(A_l \times \mathbb{F})^{\star}}_{\substack{\text{action} \\ \text{continuation}}} \} \cup \{\bot\}$$

Let *cont* denote function returning continuation for given low level action and *motiv* function returning motivation for given low level action. Utilization of motivations is inspired by the BDI [?] architecture. Agent has exactly one high level reasoning function $r_h \in A_h$ agent is also equipped in reflex reasoning function $r_r \in A_h$. Purpose of reflex reasoning function is to provide means of reacting

quickly to environment changes. Agent can execute one action at the time t, but executed action could change. This changes satisfy following conditions:

$$a_{t+1} \leftarrow \begin{cases} r_r(K(b_i, g(t))) & \text{if } r_r(K(b_i, g(t))) \neq \bot \\ a_t & \text{if } a_t(K(b_i, g(t))) = (d, \text{continue}) \\ c_1 & \text{if } a_t(K(b_i, g(t))) = (d, \text{success}) \wedge {}^{cont(a_t)=}_{(c_1,\ldots,c_n)} \\ r_h(K(b_i, g(t))) & \text{if } a_t(K(b_i, g(t))) = (d, \text{fail}) \\ r_h(K(b_i, g(t))) & \text{if } \neg motiv(a_t) \in g(t) \end{cases}$$

These conditions aren't mutually exclusive, if conflict occurs first satisfied condition is chosen.

2.2 Prolog

Prolog is a general-purpose declarative programming language, which bases strongly on a first order logic. Prolog uses notions of facts and rules, whereas execution of Prolog program is de facto a formal reasoning process. Therefore, it is very convenient to use in artificial intelligence programming [?].

Often many types of logic could be used as underlying Prolog logic, e.g. fuzzy logic [?], modal logic [?], multivalued logic [?]. Those different types of logic are used to express reasoning process in way understandable to human and all of those languages are Turing complete.

Prolog displays following properties, which make it useful in multiagent programming.

- declarative reasoning rules,
- bots knowledgebase in a symbolic form - easy to understand and analyze,
- no separation between code and data - rules could be used as data, e.g. bot leader communicating orders to another bot.

2.3 Counter-Strike Bots

Counter-Strike gameplay is a simulated environment and bots can be seen as agents within this environment. They show typical traits of agents:

- autonomy,
- making intelligent decisions,
- being mobile,
- communication and collaboration.

CS emphasises the teamwork strongly, whereas collaborative problem solving is a main domain of multiagent systems. So using this paradigm is natural in this task. It's interesting that this methodology was even earlier used in non-collaborative games like Quake with good results [?].

Actions A_l are implemented as hardcoded, but yet parametrizable, behaviors which can be acted by bots. Such behaviors may include going to a navigational

point, climbing a ladder, aiming a sniper rifle, etc. Action is description which decision at each time point does bot make.

The decision layer r_h is responsible for selecting proper actions in order to achieve bots goals. The decision layer could have different implementations e.g. behavioral networks [?], decision trees, neural networks [?], finite state machines [?] etc. In some cases decision layer could alone be decomposed into several components e.g. predictive planning, team communication, tactical navigation.

If decision layer forces bot to execute an action while it is already executing one. The old action is paused and may be resumed if new action completes.

2.4 System architecture

Counter-Strike is a mod of *Half-Life*, while CS bots are modification of the mod. *Counter-Strike* itself is not an open-source software, however it has documented ABI[1], so modifications can be created as dynamically linked libraries. Those libraries are injected into running a game and the API hooking technique is used. In this work, existing open-source bots, called *E[POD]*[2] were modified as proof-of-concept of bots programmable in Prolog. SWI-Prolog [?] was chosen as the Prolog interpreter, because it was efficient and easy to integrate into the bots code. Whole solution was programmed in Visual C++ with help of the Boost library[3]. *E[POD]* bots was programmed in object-oriented way and it already provided some objects which acted as low-level actions, however reasoning layer was programmed as FSM. In this research low-level actions were reused and reasoning layer was replaced.

SWI-Prolog offers two options of C++ and Prolog integration:

1. **C++ to Prolog** - code in C++ runs an instance of a Prolog interpreter and may run queries against it.
2. **Prolog to C++** - there could be created library of Prolog predicates which execute natively and return its results into Prolog interpreter.

Second approach is typical for development of performance-demanding libraries, which are used by Prolog later. First approach is typical for solutions, where Prolog is treated as logic engine in an application. In the bots both of these integration types where used.

The Counter-Strike engine provides routines for checking bot sensors like field of view, hearing, navigational map etc. Some of those routines were exposed into Prolog by means of native predicates. In first versions of this work bots queried Prolog engine to get the answer which actions should be executed. The code for creating an action to be executed was moved into Prolog. Rationale behind such decision was to increase a programming flexibility and code reuse. A decision to change current action can be made as result of not only logical reasoning, but also e.g. reflex or communication which are also handled by Prolog. Following

[1] Application Binary Interface

[2] `http://epodbot.bots-united.com/`

[3] `http://www.boost.org/`

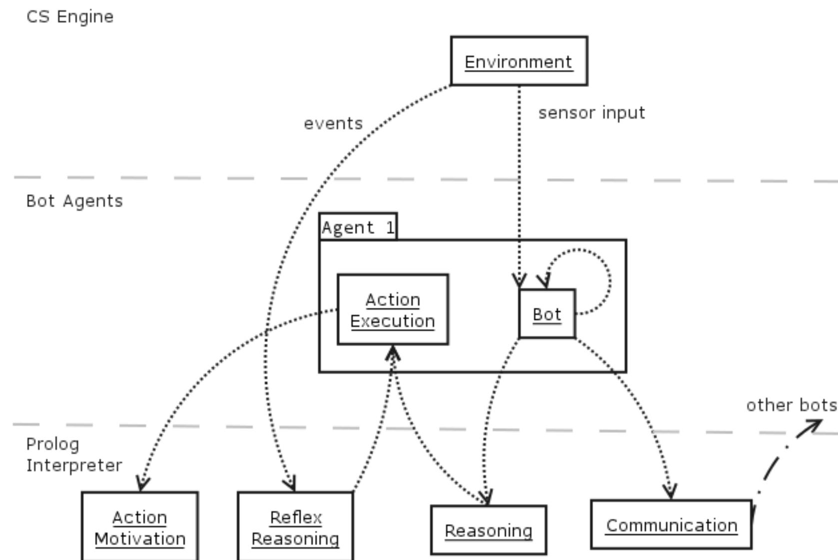

Fig. 2. Conceptual architecture diagram

enhancements were introduced to *E[POD]* bots. It will be described how formal model, presented in section 2.1, was implemented.

Motivations Motivation can be associated with each action. A motivation is a logical expression. The action is executed as long as motivation is evaluated to truth value. The motivation can be constructed by SLD-resolution mechanism. Such solution improved bot control by Prolog scripts, because script could also interrupt native action execution. Introduction of this technique improved the separation between C++ code and Prolog scripts, because script creator does not have to know detailed specification of any particular action.

Example of a motivation:

```
action_kill(
    BotID,
    EnemyID,
    and(bot_alive(Enemy), danger_low(Bot))
).
```

This action will make bot with identifier BotID try killing the bot with identifier EnemyID as long as enemy is alive and bot is not in danger.

Continuations Continuations are form of simple planning and defining complex behaviors consisting of sequence of several low-level actions. There could be assigned a continuation to any action. A continuation is a logical term which is

executed after action is successfully completed. The result of executing continuation could be new action to be executed by a bot. The new action could have its own motivation and continuation.

For example:

```
action_goto(BotID, Wp, allcost, andThen(
    action_hostages(B, andThen(
        X
    ))
)), after_free_hostages(X).
```

This action will make a bot with identifier *BotID* go to a waypoint with identifier *Wp*. The motivation for this action is the term *allcost*, which is always evaluated to truth. After reaching the waypoint bot will try to free the hostages. The action for freeing the hostages has also continuation, which is obtained by SLD-resolution and unification of variable X.

Reflex reasoning The reasoning mechanism is a deliberative process and it is computationally expensive. It runs occasionally. Bot agents also express reactive behaviors r_r to respond for rapid changes in game environment, like bot being attacked by opposing team.

Prolog script has several predicates for handling signals send directly by the game environment. Those predicates have short SLD-resolution tree in order not to impede bot performance. The predicates may cause same effect as regular reasoning process.

Communication Communication mechanism of *E[POD]* bots relayed on radio messages which was feature of CS itself. Such messages are simple imperative statements, e.g: *Taking fire, need assistance!*, *Follow me*, *Team fall back!*.

Receiving such message is handled by the reflex reasoning.

Improved teamwork One additional task was added to set of existing bot tasks written in C++. The purpose of this modification was to enhance team tactics. All bots in one team were divided into smaller groups. Each group has its own leader, who can make tactical decisions. Other bots executing additional action was blindly obeying his decisions, with some exceptions for reflex reasoning.

This behavior was introduced, because teamwork of original *E[POD]* bots was more random chance than reasonable decision - bots were acting on their own grouping only occasionally.

2.5 Examples

Example reasoning process has been shown in figure 3.

In presented example decision is made, if agent should buy new weapon. Bot should buy it, if new round has started and it is respawning, because he has

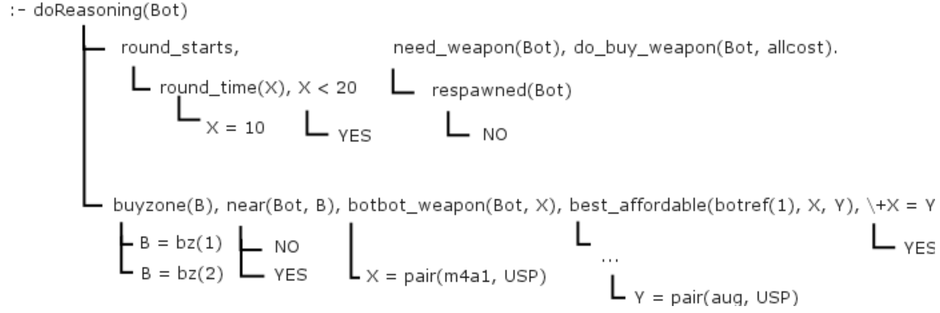

Fig. 3. Sample reasoning process

only a basic pistol. In presented example those conditions aren't meet and bot checks, if he is close to the buy zone and can afford to have better weapon than he is carrying at the moment. In this case logical formula is satisfied and bot decides to buy new weapon. If buy zone is far away it may have been wiser to play with actual weapon.

In the example in figure 4 there has been presented goal selection.

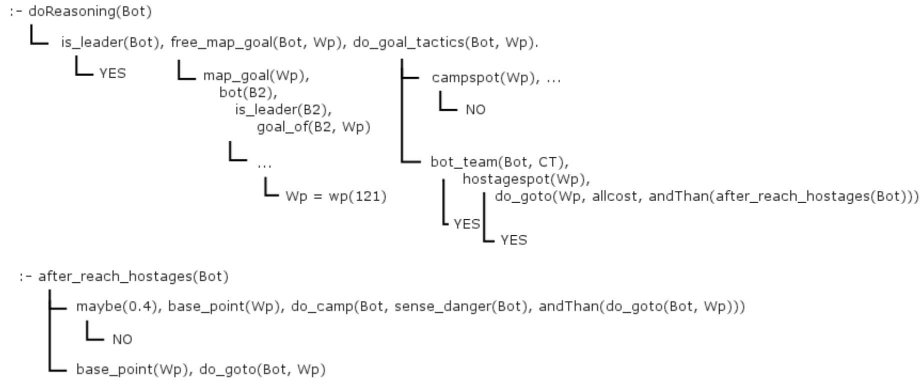

Fig. 4. Sample reasoning process involving motivations and continuations

Navigational points can represent goals for bot. Each navigational point has associated metadata e.g. if it is fit for *camping*[4]. This reasoning is carried out by team leaders. The leader choses the waypoint not chosen by any other leader. Depending on waypoint type valid tactic is chosen for execution. In presented example, there was chosen waypoint with hostages. The leader decides to move

[4] tactic involving holding static strategic position of advantage

to the hostage waypoint. This action can have it's continuation. There is an option that bot will wait near this waypoint for terrorists to come and ambush them. This action would have it's motivation - it would be executed as long as bot senses any danger e.g. hearing gunfire or enemy footsteps. In presented example bot doesn't decide to execute this action, because it is chosen with some probability. He decides to head back to base with the liberated hostages, instead.

3 Results and Conclusion

The goal of this research was to validate if it is feasible to run bots using described architecture and provide some basic formal description, which could be improved upon in course of further research. Some parts of original FSM decision layer was converted to Prolog reasoning script. Then the game was run with 32 bots - 16 for each team. Game was working with same framerate as original one, as most of the time game engine spends in idle state in order to synchronize inputs, networking, display and events in the game environment. Experiment lead to following conclusions:

1. C++ bot code can be rewritten to Prolog,
2. *Prolog bots* can be used efficiently.

More detailed performance information could be found on bar chart 5. The figure depicts percentage of total game CPU execution time taken by 3 different system components: Counter-Strike engine, bot library and Prolog reasoning layer (compare with figure 2).

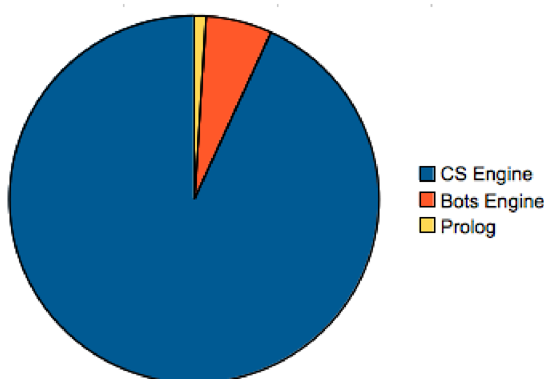

Fig. 5. CPU time consumed by different system components.

Most of a bot logic was direct translation of hardcoded C++ decision-making code into Prolog script, so it was initially expected that bots wouldn‘t play any

different than the original ones. However, new action was introduced to enhance cooperative team tactics. For that reason, additional experiments were run to verify if new bots play different than the old ones. Experiments was carried out by setting several games with different parameters setup on *de_dust* defusion map with additional action enabled for one of the teams. One parameter included sizes of groups within team which has team action enabled. Those experiments lead to conclusion that new action boosts performance of team. The optimum group consists of 3 bots including the leader. Larger group makes maneuvering task more difficult and bots tends to stuck in narrow spaces. It makes them also prone to HE grenades. Group of 2 bots does not improve gameplay that much, as it can be overpowered easier, than group of 3 bots.

Unintended, but very positive outcome was ability to test reasoning layer independently of the game engine which was very complex system. During development of Prolog scripts standard Prolog test suites were created in order to carry out regression tests. Test were easy to create because it was easy to create a mock of game engine, which wasn't started during tests at all. The game engine injected to Prolog native predicates in order Prolog script was able to interact with it. In test scenarios instead predicates created by the game engine regular Prolog predicates were declared and linked with script execution environment. Usage of this test suite allowed to reduce amount of errors leading to divergencies in expected script behavior and declared one. Improving such testing suite framework may allow to incorporate test driven development methodology (see generally [**?**]) into bots AI development.

4 Future Works

In future system will be expanded to more complete solution in order to cover more complex interactions between bots and provide rules for handling all CS map types. Also it is planned to change reasoning mechanism to one that supports basic properties of a default logic [**?**]. The 4QL language [**?**] is considered as a reasonable option. The default logic would allow to define hierarchical reasoning system in consistent way. E.g

1. general describing CS game rules and general facts, e.g
   - *if you are attacked shoot back*
2. more specific rules describing rules specific to game type, e.g
   - *on AS maps T's aren't allowed to buy AWP*[5] *rifle*
3. exceptions to the rules and special rules for concrete map
   - *on cs_assault prefer SG552 than AWP*
4. rules specific to bot
   - *if you courage is low always buy bulletproof vest before buying weapon*

[5] for firearms reference see `http://world.guns.ru/`

Such solution would allow construct modular reasoning architecture. The most specific rules and facts associated with map would be replaceable. Even if there were no rules associated with map, bots would still be able to play using general rules, but they would omit some knowledge specific to concrete map. So such map-specific rules exist to optimize bot performance on given map.

Another part of research where more improvements are going to be done is formal description of bot decision making and collaboration - at the current stage there are few discrepancies between formal description and actually implemented solution.

At some point bots were enhanced to communicate using Prolog, e.g bots could exchange messages in form of logical formulas and then deliberate. This idea was implemented, but not used. However it was not abandoned completely, because it could be developed later.

5 Acknowledgements

We would like to thank my mentor prof. Jarosław Arabas for advices provided while writing this article.

References

1. Astels, D. *Test Driven development: A Practical Guide.* Prentice Hall Professional Technical Reference, 2003.
2. Brownlee, J. Finite state machines as a control technique in artificial intelligence. Tech. rep., University of Texas at Austin, 2002.
3. Clocksin, W. F., and Mellish, C. S. *Programming in Prolog: Using the ISO Standard*, 5th ed. Springer, Sept. 2003.
4. da Silva Corrêa Pinto, H., and Alvares, L. O. An extended behavior network for a game agent: an investigation of action selection quality and agent performance in unreal tournament. In *Proceedings of the 4th Mexican international conference on Advances in Artificial Intelligence* (Berlin, Heidelberg, 2005), MICAI'05, Springer-Verlag, pp. 287–296.
5. El-Nasr, M. S., and Smith, B. K. Learning through game modding. *Comput. Entertain. 4*, 1 (Jan. 2006).
6. Gemrot, J., Kadlec, R., Bída, M., Burkert, O., Píbil, R., Havlíček, J., Zemčák, L., Šimlovič, J., Vansa, R., Štolba, M., Plch, T., and Brom, C. Pogamut 3 Can Assist Developers in Building AI (Not Only) for Their Videogame Agents Agents for Games and Simulations. vol. 5920 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, Berlin, Heidelberg, 2009, ch. 1, pp. 1–15.
7. Hagelbäck, J., and Johansson, S. J. Using multi-agent potential fields in real-time strategy games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems - Volume 2* (Richland, SC, 2008), AAMAS '08, International Foundation for Autonomous Agents and Multiagent Systems, pp. 631–638.
8. Ierusalimschy, R., de Figueiredo, L. H., and Filho, W. C. Lua - an extensible extension language. *Softw. Pract. Exper. 26*, 6 (June 1996), 635–652.

9. Jacobs, S., Ferrein, A., and Lakemeyer, G. Unreal golog bots. In *IJCAI-05 WS on Reasoning, Representation, and Learning in Computer Games* (2005), D. W. Aha, H. Munoz, and M. van Lent, Eds., Navy Center for Applied Research in Artifical Intelligence, Washington, DC, Navy Center for Applied Research in Artifical Intelligence, Washington, DC, p. 31–36.
10. Jennings, N. R., and Wooldridge, M. Applications of intelligent agents, 1998.
11. Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. RoboCup: The Robot World Cup Initiative. In *AGENTS '97: Proceedings of the first international conference on Autonomous agents* (New York, NY, USA, 1997), ACM Press, pp. 340–347.
12. Malinowski, G. *Many-Valued Logics.* Oxford Logic Guides. Oxford University Press, USA, 1994.
13. Maluszynski, J., and Szalas, A. Logical foundations and complexity of 4ql, a query language with unrestricted negation. *CoRR abs/1011.5105* (2010).
14. McPartland, M., and Gallagher, M. Learning to be a bot: Reinforcement learning in shooter games. In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2008* (2008).
15. McPartland, M., and Gallagher, M. Reinforcement Learning in First Person Shooter Games. *IEEE Transactions on Computational Intelligence and AI in Games 3*, 1 (mar 2011), 43–56.
16. Miikkulainen, R. Creating intelligent agents in games. In *Proceedings of the National Academy of Engineering* (2006), The University of Texas at Austin.
17. Morris, S. Wads, bots and mods: Multiplayer fps games as co-creative media. In *Level Up Conference Proceedings: Proceedings of the 2003 Digital Games Research Association Conference* (Utrecht, November 2003), C. Marinka and R. Joost, Eds., University of Utrecht, p. CD Rom.
18. Muñoz Hernandez, S., and Sari Wiguna, W. Fuzzy cognitive layer in robocupsoccer. In *Proceedings of the 12th international Fuzzy Systems Association world congress on Foundations of Fuzzy Logic and Soft Computing* (Berlin, Heidelberg, 2007), IFSA '07, Springer-Verlag, pp. 635–645.
19. Nguyen, L. A. The modal logic programming system mprolog. In *Proceedings of JELIA 2004, LNCS 3229* (2004), Springer, pp. 266–278.
20. Patel, P., and Hexmoor, H. Designing bots with bdi agents. In *Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems* (Washington, DC, USA, 2009), CTS '09, IEEE Computer Society, pp. 180–186.
21. Reiter, R. Readings in nonmonotonic reasoning. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987, ch. A logic for default reasoning, pp. 68–93.
22. Shoham, Y. Agent-oriented programming. *Artif. Intell. 60*, 1 (Mar. 1993), 51–92.
23. Stepnik, A. E-sport z perspektywy teorii sportu. *Homo Ludens 1*, 1 (2009), 213 – 222.
24. Tastan, B., and Sukthankar, G. R. Learning policies for first person shooter games using inverse reinforcement learning. In *Proceedings of the Seventh AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2011* (2011), pp. 085–090.
25. Vaucheret, C., Guadarrama, S., and Muñoz Hernández, S. Fuzzy prolog: A simple general implementation using clp(r). In *Proceedings of the 9th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (London, UK, UK, 2002), LPAR '02, Springer-Verlag, pp. 450–464.
26. Wielemaker, J. An overview of the swi-prolog programming environment, 2003.