

Towards an Arguing Agents Competition: Building on *Argumento*

Tangming Yuan¹ and Jenny Schulze² and Joseph Devereux³ and Chris Reed⁴

Abstract. This paper aims to open a discussion about the design and implementation of an Arguing Agents Competition, AAC - an open, competitive environment in which heterogeneous agents argue against one another. Competitors implement different argumentation strategies for the agents they submit, and can test their implementations against those of other competitors. The competition as a whole stimulates interest both in argument strategies and argumentation mechanism design in particular and automated argumentation in general. This paper shows how the infrastructure for the competition might use *Argumento*, an existing game-play engine -- but one that would require a number of enhancements to support the requirements of AAC.

1 INTRODUCTION

We hope to start a discussion in the community about the development of an open, flexible argumentation competition in which automated software can compete in various argument games. Our approach is inspired by, and could initially be modelled upon, the Trading Agents Competition, TAC [1]. The objectives of the Arguing Agents Competition are (i) to promote research in the design and implementation of arguing software agents and (ii) to apply existing public domain research and development in this area to educational and legal contexts and to social interactions.

Previous papers [2, 3] discuss a computer argument game, *Argumento*, using Dung-style argument graphs [4]. The system can facilitate human-human, human-agent and agent-agent playing argument games. A user based evaluation of the system shows that the game is both challenging and entertaining with a low learning curve. The particular interest of this paper is to discuss how to extend *Argumento* and subsequently enable it to act as a platform for an Arguing Agents Competition. We start with a brief introduction to *Argumento*, then sketch the requirements for the AAC system which lay out a number of extensions and enhancements that need to be made to the *Argumento* core. Finally, we briefly introduce some of the current examples of strategic argumentation, and how they might fit in to the AAC vision.

2 ARGUMENTO

The playgrounds for the *Argumento* games are abstract argumentation systems. An abstract argumentation system A is defined in [4] as a pair

$$A = \langle X, \leftarrow \rangle,$$

where X is a set of arguments, and \leftarrow is an attacking relation between pairs of arguments in X . The expression $a \leftarrow b$ is pronounced as “ a is attacked by b ” or “ b is the attacker of a ”.

An example of an abstract argumentation system can be seen in figure 1, which represents the pair $A = \langle X, \leftarrow \rangle$ with arguments $X = \{a, b, c, f, g, j, k, o, p, q, t, v, y\}$

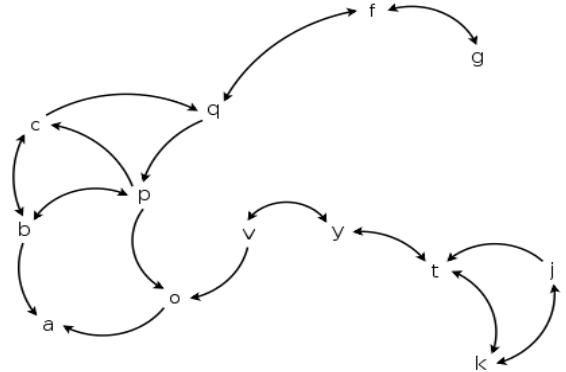


Figure 1. An example of abstract argumentation system

Argumento adopts the argument game presented in [5: 153-154; cf. 6] for reasons of simplicity. This simplicity enables players to easily follow the game rules. The argument game is formalised as a quadruple

$$G = \langle A, D, R, P \rangle,$$

where A is the argumentation system (e.g. the one outlined above), D is the dialogue history $\langle \alpha_0, \alpha_1, \alpha_2, \dots, \alpha_n \rangle$ which contains a set of moves made by game participants, R is a set of rules regulating the players' moves, and P is a pair of players $\{0, 1\}$. The set R contains six rules:

1. First move in D is made by P_0 . $P_0 = 0$

¹ University of Akureyri, Iceland, email: yuan@unak.is

² University of Akureyri, Iceland, email: HA060205@unak.is

³ University of Dundee, UK, email: josephdevereux@computing.dundee.ac.uk

⁴ University of Dundee, UK, email: chris@computing.dundee.ac.uk

2. Players take turns making moves.

$$P_i = P_{i \bmod 2}$$
3. Players cannot repeat a move.

$$\forall \alpha_i, \alpha_j \in D, \alpha_i \neq \alpha_j$$
4. Each move has to attack (defeat) the previous move

$$\alpha_i \rightarrow \alpha_{i-1}$$
5. The game ends if no further moves are possible

$$\forall \alpha_i \in A \wedge \notin D, \alpha_i \not\rightarrow \alpha_n$$
6. The winner of the game is the player that makes the final move.

$$G_{winner} = P_{n \bmod 2}$$

The latest version of *Argumento* is now part of Arg!Draw [3] which enables users to draw argumentation systems and then load them into an *Argumento* game.

Argumento is designed to enable the user to select his/her opponent. There are three choices for this: another human player, an easy agent or a smart (hard) agent. An easy agent makes a move by randomly picking a legally available argument. A smart agent has been given strategies to select the best possible arguments in order to win the game. Details of the strategies for the smart agent can be found in Section 4 below and, in more detail, in [2]. Rather than being a game player, the user can also set up two software agents and observe them playing the game. Figure 2 shows a new game configuration:

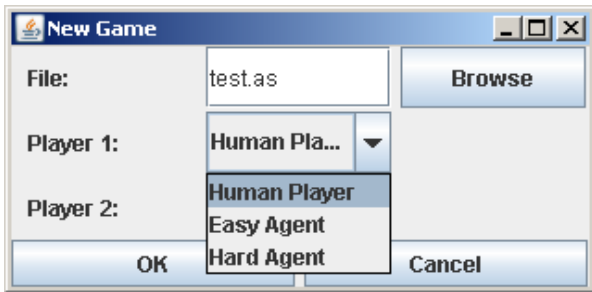


Figure 2. *Argumento* new game configurations

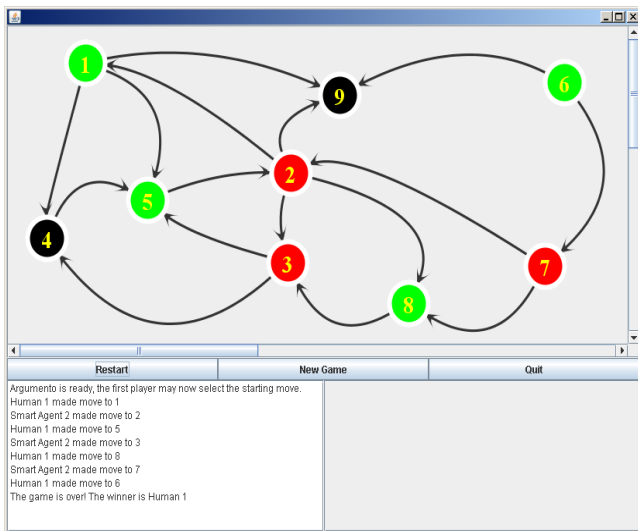


Figure 3. An example *Argumento* game

the argumentation system *test.as* loaded into the game and the possible players {human user, easy agent, hard agent}.

The example game in figure 3 shows a user playing with the smart agent. The user made the first move 1, the agent then made the only available move 2 in attacking 1. In the third turn, the user made the move 5, though it had as an alternative option 7, which also attacks 2. The game continued until the user made argument 6. In this situation the smart agent cannot locate any further arguments attacking 6, the system therefore proclaims the user the winner.

3 ARGUING AGENTS COMPETITION

The approach, then, is to use *Argumento* as the basis for an AAC, where students can investigate different dialogue strategies and construct their agents for use in the system, and subsequently for competition with each other. First it is necessary to look at the requirements of our intended AAC. The use case diagram in figure 4 illustrates this. There are two types of actors: agent programmers and AAC administrators. The agent programmers are potential students who are interested in argumentative agents and willing to develop agents to participate in the competition and the AAC administrators are those who will run the competition.

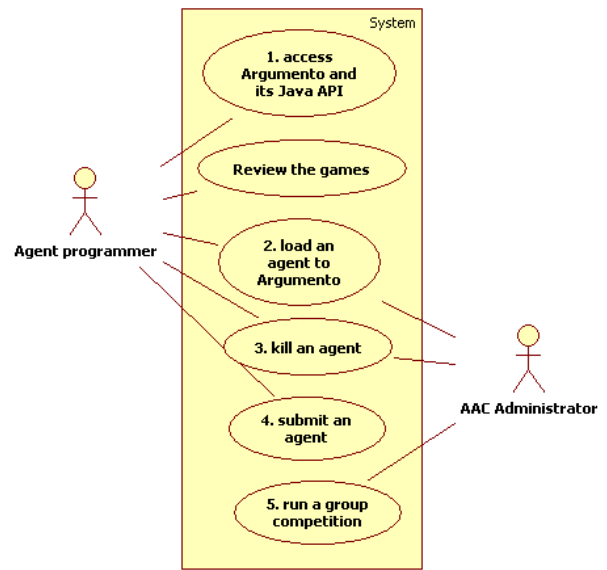


Figure 4. AAC functional requirements

A number of enhancements however need to be made to *Argumento* in order to enable it to support an AAC. They are discussed below.

The first enhancement concerns the argument game itself. The argument game outlined in the previous section does not specify the starting move. *Argumento* therefore allows players to select any available arguments as the starting move. In practice, this arrangement may result in unfair play, for example, if the first move made by the first player is not attacked by any other arguments, this will cause the second player to lose the game without making any contribution. One way to overcome this problem is to allow the first player to make an argument from a predetermined set of arguments rather than any argument in the argumentation system.

The set of potential first moves can be computed by the argumentation system component by applying the principle that every argument in the set should have the utility of 0.5. An argument with the utility of 0.5 means a player making this move has half chance of winning the game (detailed description of the utility calculation is given in Section 4 below). This responsibility can be reasonably allocated to the argumentation system. When each game starts, the argumentation system computes the set of desirable starting arguments, and then passes the set to the first player to select from. If the game had concrete arguments, desired starting moves would be the thesis or the claim of the topic under discussion.

The second enhancement concerns the enforcement of legal moves. The two agents provided by *Argumento* are programmed in a way to conform to the game rules. In cases where a human user acts as one of the players, the system will respond only when a legal move is made by the user. In an AAC setting, agents developed by different students may or may not conform to the game rules. Some extra mechanism is therefore needed to handle the situation where an agent makes an illegal move. The current proposal is to terminate the game and penalise the guilty agent with negative points (for example -2). The referee will be designed to take this responsibility. One might argue that the guilty agent should be allowed to re-make the move but this simply leaves further opportunities for the guilty agent to break the rules if it is not equipped with complex learning abilities.

The third enhancement concerns the ability to load an agent into, and delete an unwanted agent from *Argumento*. Although the current version of *Argumento* is equipped with two agents from the player's drop-down list (see figure 2), they are hard coded rather than uploaded. This feature is important for both agent programmers and AAC administrators because agent programmers need this to test and rehearse the agents they have developed and the AAC administrators need this to organise the competition.

The fourth enhancement is to enable *Argumento* to run a group competition by a single trigger. Depending on the number of participating agents, the AAC administrator can organise them into a number of groups analogous to football leagues. Each group contains (say) 3-5 agents. Within a group, each pair of agents meets twice, each of the pair starting one of their two games. The matches in the group will be played using at least three different argument graphs. The results of each game, for example the resulting argument graph, the arguing history and points awarded to both agents are to be automatically saved for later publishing. The referee component could take the responsibility of awarding points to the players (i.e. 3 points for the winner and 0 to the loser) and keeping a score sheet for the competition.

The fifth enhancement is to enable agent programmers to review the games by loading the argumentation system and the dialogue history. Games should be shown in review exactly as they were performed.

The sixth enhancement allows for *Argumento* to execute multiple dialogue games, allowing competitors to devise both strategies targeted at specific games (or classes of game) and also strategies that work well across games. This is a complex and longer-term issue, with an initial assessment carried out in [7], a companion paper.

Much has been done, and much is left to enable *Argumento* to support an AAC. Once these enhancements are made, *Argumento* and its Java API can be made accessible to the programmers (Java development kit (JDK) 1.5 or later version might be required to run *Argumento*). Agent programmers can then construct their own agents by adding *Argumento.jar* to their Java development path

and extending the *Argumento Agent* class. By extending the *Agent* class, agent programmers have to override the *makeMove()* method which returns a *Move* object. The agent class contains two substantive attributes for an agent to process in order to make a move: the argumentation system *A* and a dialogue history *D*. The argumentation system is encoded into a two dimensional boolean array, where $A_{ai, aj}$ is true if $a_i \rightarrow a_j$. Figure 5 shows an example argumentation system " $c \rightarrow b \rightarrow a$ " encoded as a 2D Boolean array, where 1 refers to true and 0 refers to false.

A	a	b	c
a	0	0	0
b	1	0	0
c	0	1	0

Figure 5. An example argumentation system encoded as a 2D Boolean array

The dialogue history *D* is structured as a *stack* of coloured arguments. The colours are consistent with their colours displayed on the graphical user interface. The top of the *stack* points to the arguments last made. The dialogue history is dynamic while the argumentation system remains static once loaded.

During the agent development process, agent programmers can test and rehearse their agents by using the facilities provided by *Argumento*, e.g., a programmer can play the game with his agent, or enable his agent to play against the dummy agents provided by *Argumento*. Upon completion, the agents submission is required to contain a text based description of the strategy of the agents, and a *.java* agent file (eg *DundeeAgent.java*). Other auxiliary *.java* files should be nested as the inner classes of the *Agent* class. Ideally submissions would be made via the Internet, but other ways, e.g. via email, would work as well.

4 STRATEGIC CONSIDERATIONS

Argumento accommodates strategic individuality, insofar as different participants may use different algorithms to choose their arguments. Thus the 'easy' agent's strategy is simply to choose at random from the legal arguments, while the 'smart' agent follows a probability-utility based strategy. This latter involves the generation of a dialogue tree on the basis of the argument graph and the opponent's first argument. The root node of the tree is the opponent's first argument, and the branches represent all legal sequences of moves, of which some will end in victory for the agent, the others in defeat. In rough terms, the utility of each node can then be generated by weighing its 'victory' branches against its 'defeat' branches. Subsequently, for each of its moves in the dialogue, the agent selects the legal argument with the highest utility [2].

While the 'smart' strategy would probably be fairly effective in most cases, it may not be ideal in all circumstances, and it would be worthwhile to devise other strategies to see how they compared to the 'random' and 'smart' strategies. It might also be interesting to modify *Argumento* by enforcing the use of argument graphs of such size and complexity, so as to render the 'smart' strategy too expensive. The time cost of this strategy is in the worst (and un-

realistic) case $O(n!)$, where n is the number of nodes in the argument graph [2], so such a modification may be worth exploring.

Such a modification would be relatively simple to implement, but it would not greatly enhance the scope for strategic individuality, as devising strategy would then be limited largely to graph-theoretic and game-theoretic considerations. Current research in argumentation envisages a richer model of strategy. As reported in [8], Moore has suggested a three-level model of strategic decision-making: (i) whether to retain or change focus; (ii) whether to build one's own view or destroy the opponent's; and (iii) the selection of methods to fulfill objectives set at levels (i) and (ii). [9], building on Moore's theory, presents a model of strategy based on an agent's *prudence* with respect to revealing its arguments and the *levels of acceptability* accorded to its arguments. Similarly, [10] presents a strategy for argumentation based on the principle of revealing as little information as possible, and [11] discusses strategic possibilities when an agent's information varies in confidentiality. Noting that information might be so highly confidential that the cost of revealing it in the course of an argumentation dialogue would outweigh the benefit accrued from winning the dialogue, the authors present a heuristic to help an agent take account of such confidentiality-related costs in argumentation.

Applying such ideas to *Argumento* is not straightforward, given the simplicity of its argumentation system and the restrictedness of its protocol. The questions of whether to change focus, and whether to build one's own view or destroy one's opponent's view cannot arise in *Argumento*; nor is the idea of information-confidentiality relevant, as a game's argument graph is completely observable to both participants.

It would therefore be desirable to modify *Argumento* so as to permit (i) the use (and perhaps creation of) argumentation systems and dialogue games besides those currently in use; and (ii) the use of standards whereby strategies can be judged with respect to such secondary considerations as information-confidentiality. A first step might be to permit the use of a dialogue game in which agents have only partial views of the argument graph. Even if it otherwise differed little from the game currently used in *Argumento*, such a dialogue game would offer new possibilities for strategic individuality.

5 CONCLUSION

We have sketched how *Argumento* might be used as the starting point for the Arguing Agents Competition. Competitors can construct agents using the Java environment, and test and rehearse their agents using *Argumento* on their own machines. At the other end, the AAC administrator can compile the submitted agents and run the competition on a stand-alone or virtualised machine to minimise the dangers presented by malware.

It would be interesting to explore a live, responsive model, in which an AAC server is always on, and games might be running continuously. Live results data might be posted online. There are some important challenges with this model, not least of which are practical, organizational issues: with games typically rather short, for example (even for large graphs, a few tens of thousands of

moves might only take a few minutes to execute), orchestrating timings becomes an issue. But the online model is attractive: that approach, its relation to *Argumento*, and the impact on protocol and architecture design are discussed in a companion paper [7].

Our immediate further work is to amend *Argumento* in line with the enhancements proposed in sections 3 and 4. Building on those beta results, and continuing the discussion with the argumentation community, we would then hope to be able to build a first fully developed AAC system by mid 2009.

REFERENCES

- [1] *Trading Agents Competition*. Available at <http://www.sics.se/tac/page.php?id=1>. [Access Date: 29 April 2008].
- [2] Tangming Yuan, Viðar Svansson, David Moore and Alec Grierson, 'A Computer Game for Abstract Argumentation', *Proceedings of IJCAI'2007 Workshop on Computational Models of Natural Argument*, 62-68, Hyderabad, India, (2007).
- [3] Tangming Yuan and Jenny Schulze, 'Drawing Tool', *Second International Conference on Computational Models of Argument (Software Demos)*, Toulouse, (2008).
- [4] Phan M. Dung, 'On the acceptability of arguments and its fundamental role in non-monotonic reasoning, logic programming and n-person games', *Artificial Intelligence*, **77** (2), 321-357, (1995).
- [5] Mike Wooldridge, *An Introduction to MultiAgent Systems*, John Wiley and Sons, New York, NY, USA (2002).
- [6] Gerard A.W. Vreeswijk and Henry Prakken, 'Credulous and Sceptical Argument Games for Preferred Semantics', M. Ojeda-Aciego, I.P. de Guzman, G. Brewka, & L. Moniz Pereria (Eds.), *Proceedings of JELIA'2000, The 7th European Workshop on Logic for Artificial Intelligence*, pp. 239-253, Springer Verlag, Berlin, (2000).
- [7] Simon Wells, Paweł Łoziński and Minh Nhat Pham, 'Towards an Arguing Agents Competition: Architectural Considerations', CMNA 2008 (under review).
- [8] Tangming Yuan, David Moore and Alec Grierson, 'A Human Computer Debating System and Its Dialogue Strategies', *International Journal of Intelligent Systems: Special Issue on Computational Models of Natural Argument*, **22**, 133-56, (2007).
- [9] Leila Amgoud and Nicolas Maudet. 'Strategical Considerations for Argumentative Agents (Preliminary Report)', *Proceedings of the 9th International Workshop on Non-Monotonic Reasoning (NMR)*, 409-17, (2006).
- [10] Nir Oren, Tim Norman and Alun Preece. 'Loose lips sink ships: a heuristic for argumentation', *Proceedings of the third international workshop on argumentation in multi-agent systems*, Hakodate, Japan, (2006).
- [11] Nir Oren, Tim Norman and Alun Preece, 'Arguing with confidential information', *Proceedings of the 2006 European Conference on Artificial Intelligence (ECAI-06)*, Riva del Garda, Italy, (2006).