Simulation League, Team *Headless-Chickens*, pages 45–48
http://www.ep.liu.se/ea/cis/1999/007/10/

An Editor for User Friendly Strategy Creation

Headless-Chickens

Johan Ydrén, Paul Scerri

JY, PS: RTSLAB, Department of Computer and Information Science, Linköping University, S-581 83 Linköping, Sweden

Abstract. *This paper describes a graphical editor designed to enable a soccer coach, without any computer science knowledge, to specify team strategies for RoboCup. The team strategy is compiled into separate agents that play according to the strategy. In an attempt to make the use of the editor natural for a coach, the design is based on the idea of a coach drawing a strategy on a whiteboard to show human players how to play during a game. The strategies that can be expressed include the formations of the players at different situations and how they move the ball within and between those formations.*

1 Introduction

In many domains where agents are used, RoboCup being one of them, there are experts who have valuable knowledge but don't have the computer knowledge to implement agents. A common way to utilize domain expert knowledge is to let the expert explain the knowledge to computer experts who in turn implements it to their understanding. This is however not a desirable solution because expert knowledge might be lost or misinterpreted in the translation and it slows development, making iterative improvement almost impossible.

This work deals with an approach to make the design of strategies for agents participating in RoboCup accessible to someone whose expertise is soccer rather than computers. Building on a method coaches use in real life a graphical editor has been designed that lets a user design a team strategy that is then compiled into separate agents.

A common way to express soccer strategies in real life is to draw them on a whiteboard using simple figures to represent the player and their actions. The design of the editor described here is based on the whiteboard concept. The editor allows a user to design team level strategies without understanding any concepts of the underlying agents. The strategies are then compiled into individual behavior based agents.

The target architecture, i.e. the architecture the agent is compiled into, is a hierarchical layered behavior based architecture. It seems to be straight forward to map the user specifications onto the architecture. The compilation process basically involves piecing predefined parts of behavioral based hierarchies together according to what has been specified in the editor.

2 The Editor

The purpose of the editor is to make the design of strategic behavior accessible to a human soccer coach whose expertise is soccer rather than computer science. A common way to express strategies in real life is to draw them on a whiteboard using simple figures representing players and arrows expressing how the players should move the ball. Using the whiteboard method a coach can express strategies at a wide range of abstraction levels. The editor described here is focused on strategies concerning the whole team and their positioning on the field in different phases of the game. Strategies aimed at structuring the flow of the game rather than low level strategies such as dribbling.

Using the whiteboard analogy, a coach expresses strategies by drawing the players at their respective positions on an image of the playing field and showing where to pass and which way to run and dribble. Figure 1 shows how a particular strategy looks in the editor. In the editor circles represents players, single line arrows represent direction to pass and double lined arrows represent direction to dribble. The strategy shown is designed to take the ball out of the teams end of the field by passing to either side and then forward.

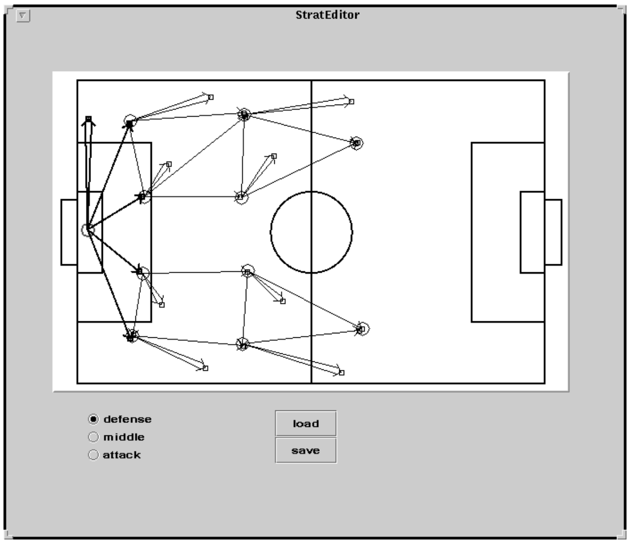

Figure 1: The editor showing a strategy intended to move the ball from the defensive zone by passing it to either side and then forward. In the lower left corner is a selector indicating the play mode

Mimicking the process the coach uses in real life, the team specification is divided into *phases* or play *modes*. Dividing a game up into simple modes is admittedly a simplification, but hopefully one that will work well for RoboCup as well as being easily understood by an user. Presently, the current ball position determines the current play mode, either defense, attack or middle when the ball is in the back, forward or center thirds of the ground respectively. For each of these modes the user assigns each player a position, a *role* i.e. attacker or defender and an arbitrary number of *actions*, i.e. dribble, pass and receive pass options. The assigned role determines how the player plays e.g. an attacker plays aggressively towards the opposite goal and a defender plays defensively and gives priority to backing up rather than moving forward. The implementation of each role corresponds to what a user would find typical for the role. The actions are integrated in the players general behavior and executed when an opportunity to do so occurs. If the user specifies more than one action for a player he can set priorities between the actions, effectively giving the agent preferred actions. Lower priority actions are chosen if higher priority ones cannot be chosen e.g. if the player can't pass to the position with the highest priority without a high probability of the opponents intercepting the ball he will chose another, better, passing action, with lower priority.

Besides making it possible to use human coaches to design strategies the editor has advantages even for computer experts. Normally designing players to act in a coordinated manner is a tedious and error prone task simply because it's difficult to visualize how players move on the field. Using the editor it's possible to view all modes at the same time and thus detect potential problems such as players being assigned to move far when the play mode changes. It is also, as can be seen in Figure 1, easy with the editor to ensure that passes will go to positions where a friendly player is likely to be.

3 Compilation

The compilation process compiles the designed strategy into eleven individual layered behavior based agents. The architecture of the agents is described in depth in [3] and closely resembles the one proposed by Blumberg [1] . Basically an agent is composed of a hierarchy of behaviors. At each level of the hierarchy one behavior is chosen to act. The choice of behavior to act is made by selecting the behavior with the highest activation, where activation is a function of the behaviors priority and the prevailing environmental conditions. At all but the lowest layer a behavior acts by setting appropriate lower level behaviors, at the lowest level behavior acts by activating a skill.

The compilation process compiles the strategy specification to a individual behavior hierarchy for each of the players. The compiler starts with a hard coded framework. The framework for each of the players is a partial generic hierarchy. The framework is the same for every player and is basically the top of the agent hierarchy. For each play mode there is a subhierarchy added to the framework. The added hierarchies implement the roles assigned to the player for the respective play modes. The hierarchies are slightly different for each role, i.e. a hierarchy for a player assigned an attacking role is slightly different to the hierarchy for a player assigned a defensive role. On to the role hierarchies are added sub hierarchies implementing each of the actions specified by the user of the editor. I.e. for every pass, dribble, or receive pass specified in the editor an action hierarchy is added to the overall hierarchy. Each of the action hierarchies is a generic hierarchy that is instantiated with parameters at the last stage of the compilation. The instantiation stage of the compilation also instantiates some parameters in the role hierarchies, for example positions to move to.

Clearly, the compilation process is quite simple, a result of how closely the architecture resembles the way a human might explain the behavior of an soccer playing agent. We consider the simplicity of the compilation process to be validation of the idea of using an underlying behavior based system rather than a simplification in the implementation. Especially helpful in the compilation process is the way behaviors, as a coach would express them, map to separate behaviors in a behavior based system e.g. when a coach uses terms like dribble or defense these behaviors are represented by separate behaviors that can be taken out or added as specified by the editors user.

4 Future work

This project is to develop a prototype which in the future may be extended in a variety of ways. Presently we are exeprimenting with integrating the high level specification system with an existing low level behavior based specification system. The user could also be given a broader range of options. An interesting area that the development of the editor has exposed is how behavior based systems can be presented to non computer scientists and how non computer scientists would find it natural to express behaviors for behavior based systems. Another interesting area is how to make a more general compiler that doesn't limit the users choices to the degree that has been found necessary in this project.

References

[1] Bruce Blumberg and Tinsley Galyean. Multi-level control of autonomous animated creatures for real-time virtual environments. In *Siggraph '95 Proceedings*, pages 295–304, New York, 1995. ACM Press.

[2] Alexander Repenning. *AGENTSHEETS: A tool for building domain-oriented dynamic, visual environments*. PhD thesis, University of Colorado, Boulder, 1993.

[3] Paul Scerri, Silvia Coradeschi, and Anders Törne. A user oriented system for developing behavior based agents. In *Proceedings of RoboCup'98*, Paris, 1998.