ChaMeleons-01 Team Description

Paul Carpenter, Patrick Riley, Gal Kaminka, Manuela Veloso,
Ignacio Thayer, and Robert Wang

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
{carpep+, pfr+, galk+, mmv+}@cs.cmu.edu
{iet, rywang}@andrew.cmu.edu

Abstract. The following paper describes the ChaMeleons-01 Robocup 2001 simulation league team. Development was concentrated in two main areas: the design and implementation of an action-selection architecture and the development of the online coach. The architecture was designed in such a way to support the integration of advice from an external agent. Currently, the only external agent the ChaMeleons-01 support is an online coach offering advice using the standard coach language. The online coach offers advice in the form of passing rules, marking assignments, and formation information.

1 Introduction

The ChaMeleons-01 team is a new team developed for the Robocup 2001 simulated robotic soccer competition. Although two members remain from past Carnegie Mellon simulation teams, the addition of new team members and the desire to pursue new research directions prompted the decision to start a team from scratch. The ChaMeleons-01 coding effort started with the world model from the CMUnited-99 publicly released source code. Although some time was spent on the skills development, most of the effort was focused on building a team that is highly coachable and able to easily adapt to different opponents. Implementing a coachable team requires not only the development of a coach but also considerations in the design of the team with regards to how advice will be integrated into the action selection process. In section two, the architecture designed and implemented to support the coach will be described and in section three, the implementation of the coach will be described.

2 Action-Selection Architecture

The ChaMeleons-01 use a hierarchical, behavior-based architecture. The major feature of the design is that it facilitates the addition of advice from outside sources. This is accomplished through the use of a Behavior Manager (BM) and different Behavior Arbitrators. Presently, the only advice the players use is from

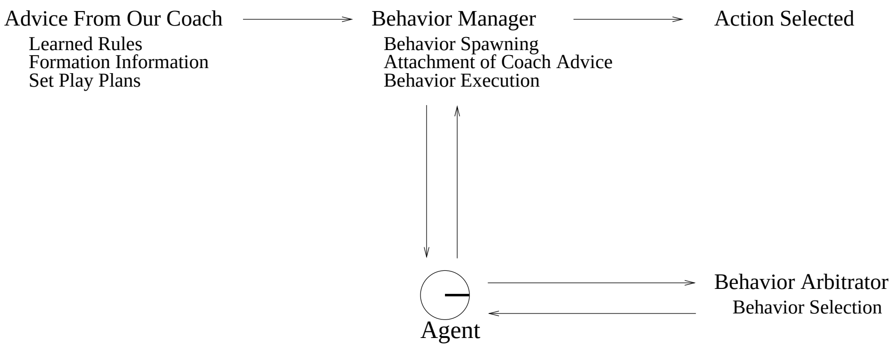

Fig. 1. Action Selection Architecture

the online coach but other players or human operators could potentially provide advice. Figure 1 shows how each of these components are related.

A behavior generates all of its children when it begins execution. For example, every time *passBall* executes, it generates a *passToPlayer* behavior for every teammate whose position is known. If a behavior from the list of coach advice matches one of the children types of the executing behavior, the BM attaches it to the list of children.

All behaviors inherit from the same abstract behavior class. Each behavior provides several methods to aid in the action selection process. Most of the agents decision making rely on a behavior's *isApplicable* method. For example, the *handleBall* behavior is applicable only if the ball is kickable. Other methods include *isChildOf* which returns whether or not a given behavior has the same type as one of its children and *probabilityOfSuccess* which estimates the probability that a behavior will execute successfully. Behaviors range from the lowest level behaviors like *dash*, *kick*, and *turn* to higher level behaviors such as *dribbleBall*, *passBall*, and *shootBall*.

The BM creates and executes behaviors while also attaching the matching coach advice when appropriate. Because players will tend to consider a lot of the same actions from one cycle to the next, the BM maintains a pool of the most recently used behaviors. When a behavior attempts to execute another behavior, the call goes through the BM. The BM first checks if the behavior currently exists in memory before it allocates a new behavior. The BM logs each request to execute a behavior and whether or not the behavior executed successfully. Logging the entire call chain aids in locating bugs while debugging.

The Behavior Manager also integrates all advice from the coach. For an executing behavior, the BM searches through the coach advice and adds to the list of children any behaviors that match one of the children types. For example, if a coach recommends that player seven pass to player ten, when player seven begins to execute the *passBall* behavior, the action recommended by the online coach, *passToPlayer* corresponding to pass to player ten, will be added to the list of children it considers. The BM also flags the behavior as originating from the pool of advice behaviors. It is important to note that simply because a coach offers advice to a player, there exists no obligation on the player's part to follow the advice. A behavior arbitrator ultimately decides among the considered behaviors.

A behavior arbitrator selects which behavior to execute when more than one behavior is applicable. The arbitration method can be any heuristic and it does not have to be the same for all behaviors. Simple heuristics include executing the first behavior it considers that is applicable or always executing coach advice before considering anything else.

The *handleBall* behavior uses a much more complex arbitration scheme. In this case, behaviors are classified by descriptors such as *passForward*, *passBack*, *passToMostOpen* with each descriptor having its own priority. Behaviors that could potentially result in a goal have the highest priority, e.g., shooting the ball, passing to someone that has a great shot, while other behaviors have lower priorities. The prioritization is currently done by hand but it is feasible to learn the priorities using machine learning techniques.

Whether or not to actually execute a behavior or to consider the behavior at the next highest priority level depends on the behavior's probability of success. Each priority level contains a threshold for the probability of success; if the threshold is not met, the next highest level is considered.

The *handleBall* arbitration method can also determine whether or not to enable a particular behavior that has not met the required threshold. For example, if a player has the ball near the opponents goal and the probability of success for a shot is just below the threshold, the best thing to do may be to hold the ball for a few cycles to see if a better shot develops. Although the capability of enabling behaviors exists, it was not used in Robocup-2001.

3 Online Coach

The ChaMeleons online coach (also known as OWL) competed in the online coach competition as well as providing coaching for the ChaMeleons team during the team competition. Besides substitution of heterogeneous players, the coach generates advice using the standard coach language as described in the Soccer Server Manual[5].

OWL uses a simple heuristic approach to choose heterogeneous types. Based on empirical data gathered in a fixed training environment as well as the role of the agent, the coach evaluates the player types on their kicking ability, interception ability, and affect on stamina. The player types with the highest heuristic value are substituted for the default players. All substitutions are made at the beginning of the game.

As far as the opponent's types, the coach carefully tracks error bounds in order to evaluate whether each opponent agent's actions (dashes and kicks) are consistent with a particular type. If a type is not consistent, that information is transmitted to the player through 'info' messages in the standard coach language.

Similar analysis is done based on the player size, since (in normal cases), two object cannot overlap.

Based on the setplays planning developed for ATTCMUnited2000[2], the coach creates a plan for ball and player movement for each setplay situation. The primary difference is that the plan is expressed as a set of condition-actions rules using the standard coach language. Further information about setplays can be found in [3, 4].

OWL also has the ability to learn formations based on observations of a team playing. The learning is done in two stages. First, for each player, a small rectangle is found which encloses most of the points in which that player was throughout the game. Then the rectangles are shifted in a hillclimbing search on the field. The evaluation function favors positions where the average distance between the rectangles for players are close to the same as the average distance observed throughout the game.

OWL can then send this formation to the team using the 'home' action of the standard coach language. In addition, if OWL is given the expected formation for the opposing team, OWL can assign marks to the defenders.

The final important module of the coach deals with learning to imitate another team's passing behavior. First, a team to imitate is chosen by hand. In the online coach competition, we chose to imitate Brainstormers as they played against Gemini. OWL extracts all of the successful passes made by both teams when playing each other. Then, using C4.5, rules are extracted to predict where a pass will go to. The input conditions are the starting location of the pass and the distance and angle of all players. Note that the starting and ending locations of the passes are discretized using Autoclass C[1].

These rules are then translated into coach language advice in the following fashion. For the team to be imitated (Brainstormers for the coach competition), the rules are translated into 'pass to region' actions with the appropriate conditions. For the opponent (Gemini for the coach competition), the rules are translated into 'mark line region' directives instructing the team to try and block the predicted pass.

References

1. P. Cheeseman, J. Kelly, M. Self, J. Stutz, W. Taylor, and D. Freeman. Autoclass: A bayesian classification system. In *ICML-88*, pages 54–64, San Francisco, June 1988.
2. Patrick Riley, Peter Stone, David McAllester, and Manuela Veloso. ATT-CMUnited-2000: Third place finisher in the RoboCup-2000 simulator league. In Stone, Balch, and Kreatzschmarr, editors, *RoboCup-2000: Robot Soccer World Cup IV*. 2001.
3. Patrick Riley and Manuela Veloso. Planning for distributed execution through use of probabilistic opponent models. In *IJCAI-2001 Workshop PRO-2: Planning under Uncertainty and Incomplete Information*, 2001.
4. Patrick Riley and Manuela Veloso. Recognizing probabilistic opponent movement models. In *Proceedings of the RoboCup International Symposium*, 2001.
5. RoboCup Federation, http://sserver.sourceforge.net/. *Soccer Server Manual*, 2001.