Constructing Game Agents from Video of Human Behavior

Nan Li[1], David J. Stracuzzi[1], Gary Cleveland[1], Tolga Könik[2],

Dan Shapiro[2], Matthew Molineaux[3], David Aha[4], Kamal Ali[2]

[1]School of Computing and Informatics, Arizona State University

[2]Computational Learning Laboratory, Stanford University

[3]Knexus Research, Springfield, VA

[4]Naval Research Laboratory, Code 5514, Washington, DC

Abstract

Developing computer game agents is often a lengthy and expensive undertaking. Detailed domain knowledge and decision-making procedures must be encoded into the agent to achieve realistic behavior. In this paper, we simplify this process by using the ICARUS cognitive architecture to construct game agents. The system acquires structured, high fidelity methods for agents that utilize a vocabulary of concepts familiar to game experts. We demonstrate our approach by first acquiring behaviors for football agents from video footage of college football games, and then applying the agents in a football simulator.

Introduction

Building agents that act intelligently in computer games is an important task in game development. Traditional methods for encoding game agents are time-consuming, and the resulting behaviors are often not very flexible. Cognitive architectures (Newell 1990), which are designed to model human-level intelligence, are a relatively recent addition to the game developer's toolkit in constructing autonomous agents. They have the advantage of providing mechanisms for reasoning and decision-making, so that the game developers need only specify static domain knowledge. Moreover, the architectures can acquire any missing or hard-to-specify knowledge through domain experience and observation of other agents.

In this paper, we apply the ICARUS cognitive architecture to the problem of constructing flexible game agents for a football simulator. The system takes in discrete perceptual traces generated from video footage of college football games, analyzes the traces with provided background knowledge, and learns complex behaviors that the agents can then apply in a simulated football game. The acquired knowledge is composable, which allows ICARUS to generate team behaviors not observed in the video, and is encoded in a human interpretable format, which allows for subsequent modification by game developers.

We begin by introducing the football videos observed by ICARUS, along with the Rush 2008 football simulator, which we use to demonstrate our approach. We then review the ICARUS cognitive architecture, including recent work on acquiring domain knowledge from observed human behavior. Next, we demonstrate and evaluate the behaviors acquired and executed by ICARUS. Finally, we discuss the impact of this work on the game-agent construction task, and conclude with a summary of related work and directions for future development.

From Video to Game Agents

The specific task that we consider in this paper is to acquire football agent behaviors by observing videos of several college football plays. The learned behaviors should have high fidelity with respect to the original videos, and should exhibit utility consistent with the game of football when executed in the simulator. We first preprocess the raw video data into sequences of ICARUS perceptions, which the architecture uses to acquire game agent behaviors. We then test the ICARUS agents in the Rush simulator by comparing against hand-crafted agents written in the simulator's own play-design language. In the remainder of this section, we describe the video which our method will observe, and the simulator in which we will test the acquired behaviors.

The College Football Video

The raw videos used in our experiments depict individual plays as executed by the Oregon State University football team. The video was shot via a panning and zooming camera that is fixed at the top of the stadium. The video corresponds to that used by coaches, and the camera operator attempts to keep as much of the action in view as possible. A typical shot from our video is shown in Figure 1.

ICARUS does not possess any specialized mechanisms for handling visual data. We therefore convert preprocessed versions of the videos into a sequence of ICARUS perceptions for the architecture to observe. Specifically, the perceptual representation includes: (1) the 2D field coordinates of each player at each video frame (Hess and Fern 2009), (2) player labels describing the functional role of each player (e.g. quarterback, running back, etc), and (3) activity labels describing the low-level activity (such as running or blocking) of each player throughout the play. We used the activity labels and player labels generated by (Hess, Fern, and Mortenson 2007) to produce the perception sequence for the play.

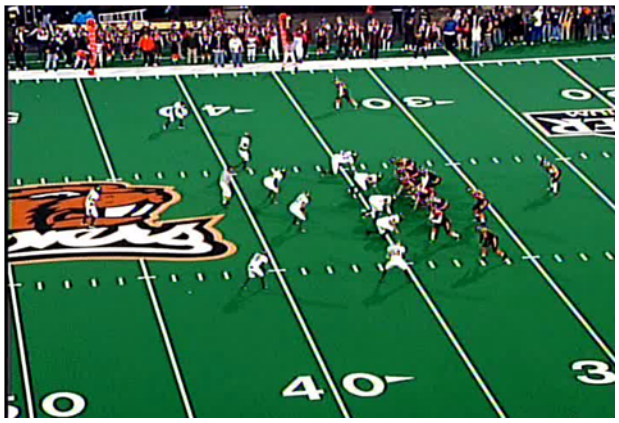

Figure 1: A typical frame from the football video.

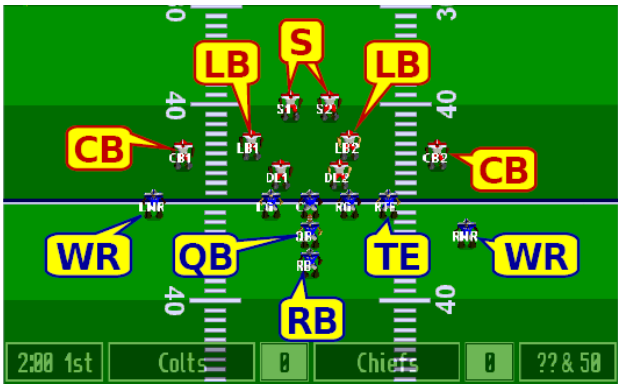

Figure 2: A Rush 2008 starting formation for the offensive (bottom) and defensive (top) teams, with player/position annotations.

The Rush 2008 Football Simulator

Rush 2008, a research extension to Rush 2005,[1] simulates an eight player variant of American football (which typically has eleven players per team). Figure 2 shows a typical starting formation for the simulated plays. The rules and objectives in Rush are similar to those in American football. Each player is assigned a role, such as quarterback (QB) or running back (RB), and can be controlled by a set of instructions. Players are controlled either by assigning them a high-level goal such as *pass route cross out at yard 15* (which instructs a receiver to run 15 yards down-field, make a hard right turn, and then keep running to try to catch a pass), or by assigning specific instructions such as *stride forward* on each clock tick. The hand-coded plays in our experiments use the former types of instructions to take advantage of the expertise built in to the simulator, while ICARUS uses the latter form to demonstrate the architecture's ability to construct such knowledge on its own.

We instrumented Rush so that ICARUS can perceive all players and objects on the field and control the actions of each offensive player on a tick-by-tick basis. Each offensive player shares perception and knowledge with the other offensive players, and carries out actions selected by ICARUS. Example actions include *(throwTo <receiver>)*, which instructs the QB to pass to a specific teammate, and *(stride <direction>)*, which tells a player to run in one of eight directions for one clock tick. Defensive players are controlled by the simulator, which randomly selects one of several available strategies for each play.

A Brief Review of The Icarus Architecture

ICARUS (Langley and Choi 2006) shares many features with other cognitive architectures like Soar (Laird, Rosenbloom, and Newell 1986) and ACT-R (Anderson 1983). For example, all three architectures distinguish between short-term and long-term knowledge bases, and provide goal-driven but reactive execution. ICARUS also includes several novel features, such as separate storage for conceptual and skill (behavioral) knowledge, and skills (behaviors) indexed by the goals they achieve. In this section, we briefly review key aspects of the framework to provide background for the experiments and demonstration.

Beliefs, Concepts and Inference

One of the principal tasks that an intelligent agent performs is reasoning about its environment. This is necessary to determine which actions the agent must carry out to achieve its goals. ICARUS performs this inference task by matching the conceptual structures in its long-term storage against percepts and beliefs stored in short-term knowledge bases.

ICARUS operates in cycles that begin when an agent receives low-level perceptual information about its environment. Percepts typically describe the attributes of a single entity (such as a player), and are very short lived. For example, the percepts derived from the college football video last only for the duration of one video frame ($1/30^{\text{th}}$ of a second) before being replaced with new information.

Intelligent behavior requires more than low-level perceptual information. ICARUS therefore includes an episodic belief buffer (Stracuzzi et al. 2009) that contains higher-level inferences about the environment. Beliefs represent relations among entities, and have two associated time stamps. The first indicates the first time at which the belief held, while the second indicates the last time at which the belief held continuously. All beliefs inferred during the current episode are retained in the belief buffer, allowing the agent to reason about events over time.

ICARUS beliefs are instances of generalized concepts stated in a hierarchically organized, long-term conceptual knowledge base. Each concept has a head, which consists of a predicate with arguments, and a body, which defines the situations under which the concept is true as shown in Table 1. The relations field specifies the subconcepts a concept depends upon along with their associated time stamps which correspond to the time stamps on beliefs. The constraints field then describes the temporal relations among the subconcepts using these time stamps. For example, the concept *possession* aggregates and stores perceptual information about the ball carrier into belief memory (symbols pre-

[1] http://rush2005.sourceforge.net/

Table 1: Sample concepts for the football domain.

```
; Indicates that ?agent carried ?ball in the current time step
((possession ?agent ?ball)
  :percepts     ((ball ?ball carriedby ?agent)))

; ?passer dropped back ?n-steps after receiving the snap
((dropped-back ?passer ?n-steps)
  :relations    (((snap-completed ?passer ?ball) ?snap-start NOW)
                 ((possession ?passer ?ball) ?poss-start ?poss-end)
                 ((moved-distance ?passer ?n-steps S)
                   ?mov-start ?mov-end))
  :constraints ((≤ ?snap-start ?poss-start)
                 (≤ ?mov-end ?poss-end)
                 (= ?mov-end NOW)))
```

Table 2: Sample skills from football.

```
; skill for running downfield ?dist yards, then turning and
; running in ?dir until the ball is caught (by any agent)
((cross-pattern-completed ?agent ?dist ?dir)
  :start       ((sccross-pattern-completed-c2 ?agent ?dir))
  :subgoals ((moved-distance-in-general-direction ?agent ?dist N)
              (moved-until-ball-caught ?agent ?dir)))

; skill for running a cross pattern, then catching and
; running with the ball
((cross-reception-completed ?agent ?dist ?dir)
  :subgoals ((cross-pattern-completed ?agent ?dist ?dir)
              (ran-with-ball-until-tackled ?agent ?ball)))
```

ceded by a question mark denote variables). Similarly, the constraints field of *drop-back-completed* states that *?passer* must have the possession of *?ball* until he has finished dropping back. Note that concepts can only describe properties of states. They do not contain information about how to achieve these properties.

ICARUS updates the episodic memory at the start of each cycle by matching the concept definitions to percepts and existing beliefs in a bottom-up manner. After the percepts are updated, low-level concepts (such as *possession*) are matched against them and the results are stored as beliefs. This then triggers matching against higher-level concept definitions. The process continues until the deductive closure of percepts, beliefs and concepts has been computed.

Goals, Skills and Execution

After inferring a set of beliefs about its environment, ICARUS then evaluates its skill knowledge (behavioral methods) and decides which actions to take. This is how the architecture controls behavior. Toward this end, the architecture uses a *goal buffer*, which includes goals the agent wants to achieve. It then retrieves behaviors capable of achieving these goals from the long-term skill knowledge base.

Like concepts, the skills are hierarchically organized and are indexed by the concepts defined in the conceptual knowledge base. Each skill consists of a head, which specifies the goal the agent achieves after carrying out the skill, a set of start conditions, that must be satisfied to initiate the skill, and a body, which states the steps the agent should follow to achieve the goal. For example, Table 2 shows the method for completing a cross-pattern, which requires the agent to first run downfield (north) for *?dist*, then turn in *?dir* (east or west) and run until the ball is caught. *Cross-reception-completed* builds on this by having the agent run north with the ball until it is tackled.

Given a goal and the knowledge base of methods, execution proceeds by first selecting an unsatisfied top level goal, and then finding methods that are known to achieve the goal. Since one behavior may refer to others, the system will find an executable path through the skill hierarchy which terminates with a low-level behavior that corresponds to an action that the agent can execute directly in the environment. ICARUS continues executing along this path until the goal is achieved or the selected skills no longer apply, in which case it must determine a new skill path from the current state to the selected goal.

Learning Skills from Video

Manual construction of behaviors, particularly in the context of temporal constraints, remains time-consuming. In previous work, Li et al. (2009) integrated a learning method that acquires hierarchical skills from problem solutions into ICARUS. The input consists of a goal, a set of concepts sufficient for interpreting the observed agent's behavior, a set of low-level methods available in the environment, and a sequence of observed perceptual states. Optionally, any known or previously acquired methods may be included.

The algorithm runs in three steps. First, the system observes the video of the game and infers beliefs about each state, storing the results in belief memory as described above. Next, the agent explains how the goal was achieved using both existing conceptual and procedural knowledge. The agent first tries to explain the trace with known skills that both achieve the goal and are consistent with the observations. It then selects one of the candidate skills and parses the observation trace into subtraces based on the times when the start conditions and subgoals were achieved.

If the agent fails to explain the trace with procedural knowledge, it then attempts to use concepts. First, it retrieves from memory the belief associated with the goal, including the lower-level beliefs that support the goal belief. Then, it parses the observation sequence based on the times when these subbeliefs became true. This explanation process continues recursively until the agent builds explanations that show what sequence of events and/or known behaviors led the agent to achieve its goal. These then form the basis for learning new behaviors. Note that new methods are constructed on top of existing methods based on the derived explanations. Li et. al (2009) provide a more detailed discussion of the learning algorithm.

The algorithm for acquiring methods is not equivalent to compiling information already contained in the provided concept hierarchy into an operational form. The learning mechanism must recognize which parts of a concept definition are start conditions, and which are subgoals. If we simply consider all subconcepts in a definition to be subgoals, then ICARUS will believe that the learned behaviors apply in

many situations that they do not. The result would be that agents take many fruitless actions in the environment. Similarly, the temporal constraints in a concept definition only provide partial ordering information about the subgoals in a new method. Although the concepts in football tend to provide a total ordering over the subgoals, this is not generally true, and our approach does not rely on this property.

Demonstration and Evaluation in Rush 2008

Our objectives in evaluating the learned skills are to show that they have both high fidelity with respect to the original video, and utility similar to the observed play with respect to the game of football. To evaluate the former, we performed a qualitative comparison of the plays executed by ICARUS and Rush to the original videos. For the latter, we compared the yardage gained by the learned agents to yardage gained by the hand-constructed agents for the same play.

Three steps were required to acquire the skills from video and then execute them in Rush. First, the raw video was preprocessed to produce a sequence of ICARUS perceptual states. Second, we applied the learning system to observe the preprocessed sequence only once and construct skills based on the observation. Finally, we mapped the learned eleven-player skills onto an eight-player team, and executed the resulting skills in Rush. We have discussed the first two steps in previous sections. Below we provide details for mapping plays into Rush, along with the results.

Applying ICARUS and Mapping Plays into Rush

As noted, a team in Rush consists of eight players while the teams from the video have eleven players. We therefore map the eleven-player skills learned from the video into skills for the eight Rush players. In practice, we can accomplish this simply by dropping the goals associated with three of the players. Ideally, we would ignore the players that have the least impact on the play.

Figure 3 shows a play diagram for play 18, observed by ICARUS during testing. Rush plays typically include three offensive linemen (LG, C, and RG) along with some combination of four running backs and receivers. The college play shown in Figure 3 has five linemen and five backs/receivers. To map this play into Rush, we therefore dropped two linemen (LT and RT) and one running back (RB), all of which had top-level goals of pass-blocking for the duration of the play. ICARUS used the goals for the eight remaining players to guide execution in Rush.

Evaluating Plays in Rush

To measure the quality of the constructed agents, the plays executed by both ICARUS and the hand-coded Rush agents were evaluated according to several qualitative measures, along with average yardage gained by the plays. We allowed ICARUS to observe three distinct passing plays (13, 18, and 63) for learning purposes. The hand-coded agents were created by a Rush expert. Both sets of agents were run 10 times on each of the three plays in Rush to help factor out random differences in the behavior of the defense.

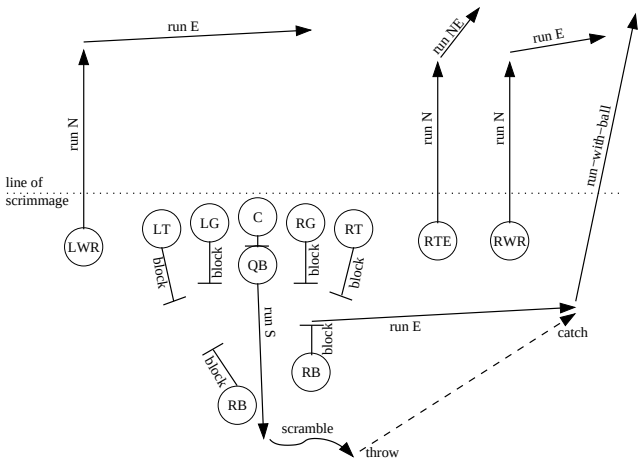

Figure 3: Diagram of play 18 with annotations indicating actions taken by individual players.

The resulting plays were evaluated according to six qualitative measures, three of which have associated yardage measures. The qualitative measures indicate a percentage of the 10 test plays in which the desired behavior appeared, while the numeric values show the average yardage gained on the subset of plays that meet the qualitative criterion. *Pocket* refers to the behavior of the three offensive linesmen, who should block for the quarterback while he seeks his pass receiver. *Receivers* refers to the behavior of the running backs, wide receivers, and tight ends in the play, and is only satisfied if *all* of the receivers successfully complete their assigned pass patterns. *Interceptions* are undesirable events that occur when a defensive player catches the ball thrown by the quarterback. The numeric result for an interception is zero yards. *Unintended* refers to the case in which the quarterback throws the ball to an offensive player other than the intended receiver. This is less desirable than passing to the intended receiver, but may still net positive yardage. *Intended* refers to the case in which the quarterback completes a pass to the intended receiver (as determined by the videos). *Team* indicates cases in which all players on the team successfully achieved their respective goals. Associated yardage indicates average yards gained over all 10 plays, regardless of how many plays actually succeeded.

Results

In all three plays, the learning system successfully captured the play patterns associated with all offensive players. Note that since our learning mechanism relies on relational knowledge, it captures general skills from the noisy perceptual sequences generated from the videos provided that the noise does not render portions of the play unobservable.

Table 3 shows the results of testing both the ICARUS and hand-constructed agents on all three of the observed video plays. In play 13, shown in Figure 4, ICARUS outperforms the hand-crafted agents along all dimensions, gaining more yards per attempt, and generating more of the behavior found in the video. Notably, the hand-crafted agents failed

Table 3: Qualitative and quantitative test results for both ICARUS and hand-crafted agents.

	pocket	*receivers*	*interceptions*	*unintended*	*intended*	*team*
ICARUS-13	100%	100%	0%	40%, 1.3y	40%, 12.9y	40%, 5.7y
Hand-13	100%	10%	20%	0%	50%, 10.5y	10%, 5.3y
ICARUS-18	100%	0%	0%	0%	100%, -0.9y	0%, -0.9y
Hand-18	100%	100%	0%	0%	100%, 2.3y	100%, 2.3y
ICARUS-63	100%	100%	0%	0%	100%, 6.1y	100%, 6.1y
Hand-63	100%	100%	0%	0%	100%, 9.9y	100%, 9.9y

to have the inner wide receiver, LWR, complete his slant passing route in several of the plays because the intended receiver caught the ball before LWR could turn toward the northwest. This indicates a possible discrepancy between the speed of the receivers in the video and the simulator because this particular receiver route was longer than the others. Also noteworthy is the relatively high rate of reception by an unintended receiver for ICARUS. This is caused by a combination of the play design, which has two receivers crossing the ball's trajectory at approximately the same time, and again differences in simulated and real player speeds. In the original play, there was no ambiguity as to who could or should catch the ball. Both issues suggest that additional fine-tuning of the play in the simulator may be helpful.

ICARUS was less successful at navigating play 18, shown in Figure 3. The average yardage gained was negative in this case, although this is not unreasonable behavior for a short pass behind the line of scrimmage. The hand-crafted agent gained an average of 2.3 yards per attempt, suggesting that some fine-tuning of the ICARUS play could improve performance. ICARUS also failed to successfully complete its receiver patterns. In this case, all of the failures stemmed from a single error. The architecture assigned too deep of a pattern to the tight end, RTE, rendering him unable to complete his pattern before the receiver caught the ball. This is another case in which fine-tuning of the play in the simulator could improve performance noticeably. All other receivers executed successfully.

In play 63, both approaches performed well. On average, ICARUS tended to gain fewer yards than the hand-crafted agent, but closer inspection of the video and resulting agents suggests that this is because ICARUS is more faithful to the video than the hand-crafted agents. Specifically, the intended receiver, RWR, runs a short crossing pattern (five yards north, followed by a hard cut to the west). ICARUS duplicates this exactly, but the hand-crafted agent runs diagonally northeast until he catches the ball, ultimately putting him further up-field than the ICARUS agent. Such a maneuver may not have been possible in the original 11-player game, as the additional players may have impeded the receiver's progress. This also suggests that additional learning in the simulator may be fruitful.

Related and Future Work

Constructing autonomous agents is an important task in video game development. Games such as Quake, Warcraft III, and Halo 2 (Damian 2005) use finite state machines to control autonomous agent. However, the complexity of fi-

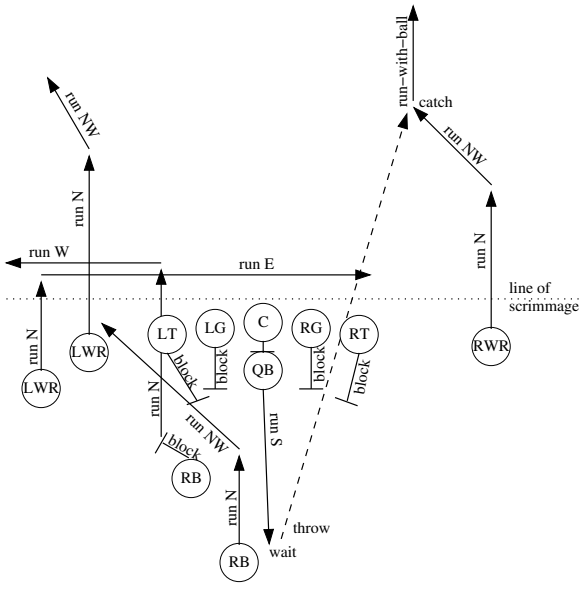

Figure 4: Diagram of video play 13 with annotations indicating actions taken by individual players.

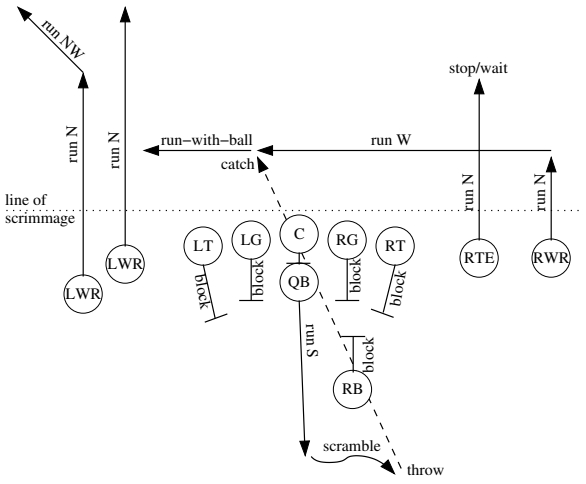

Figure 5: Diagram of video play 63 with annotations indicating actions taken by individual players.

nite state machines, hierarchical or otherwise, increases dramatically with the complexity of the game (Orkin 2006). Scripts are another approach to modeling agents used for example in Neverwinter Nights and Unreal Tournament. As with state machines, script complexity increases quickly as agent behavior becomes more complicated. Scripts are also typically composed offline, leading to less flexible agents. Our work differs from these in that the cognitive architecture provides a higher-level language with which to model autonomous agents.

The use of artificial intelligence in games is a recent development aimed at reducing the effort required to construct agents. For example, Kelly et al. (2008) use offline planning with hierarchical task networks to generate scripts automatically. This approach requires expert knowledge to build these hierarchical task networks, while our system acquires similar knowledge automatically. An alternate approach uses reinforcement learning algorithms (Bradley and Hayes 2005; Nason and Laird 2005) to allow an agent to learn through experience. Our approach differs because learning is analytic and based on observed video rather than agent experience. This makes our approach much more computationally efficient.

There are many possible avenues for future development of this work. First, the current learning system focuses on learning from single agent behavior. However, football is a multi-agent game with sophisticated coordination among agents. Ideally the system should learn knowledge about timing and cooperation among multiple agents. Second, the skills and plays acquired from the observed video can also be further polished by learning within the Rush simulator, as noted in the context of all three demonstration plays. This can be as simple as modifying the arguments given to the learned methods, or as complex as modifying the structure of the learned behaviors. Finally, the current learning system relies on a concept hierarchy. We are in the process of developing concept learning mechanisms which, once integrated with ICARUS will reduce the amount of expert knowledge required by the system in the form of concepts.

Conclusion

Constructing autonomous agents is an essential task in game development. In this paper, we outlined an approach to acquiring complex agent behavior based on observation of video footage of college football games. We showed that our cognitive architecture-based approach produced agents competitive with hand-engineered agents, and provided several possible next steps to further improving the quality of the learned agents. Our results suggest that using architectures for intelligent agents, such as ICARUS, is a viable and efficient approach to constructing intelligent game-agents that utilizes a domain-relevant vocabulary, and reduces the amount of required expert design and knowledge.

Acknowledgments

This material is based in part on research sponsored by DARPA under agreement FA8750-05-2-0283. The U. S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies or endorsements, either expressed or implied, of DARPA or the U. S. Government.

References

Anderson, J. 1983. A spreading activation theory of memory. *Journal of Verbal Learning and Verbal Behavior* 22:261–295.

Bradley, J., and Hayes, G. 2005. Group utility gunctions: Learning equilibria between groups of agents in computer games by modifying the reinforcement signal. In *Proceedings of the IEEE Congress on Evolutionary Computation*, 1914–1921. Edinburgh, UK: IEEE Press.

Damian, I. 2005. Handling complexity in the Halo 2 AI. In *Game Developers Conference*.

Hess, R., and Fern, A. 2009. Discriminatively trained particle filters for complex multi-object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. Miami, FL: IEEE Press.

Hess, R.; Fern, A.; and Mortenson, E. 2007. Mixture-of-parts pictorial structures for objects with variable part sets. In *Proceedings of the Eleventh IEEE International Conference on Computer Vision*. Rio de Janeiro, Brazil: IEEE Press.

Kelley, J. P.; Botea, A.; and Koenig, S. 2008. Offline planning with hierarchical task networks in video games. In Darken, C., and Mateas, M., eds., *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference*. Stanford, CA: AAAI Press.

Laird, J. E.; Rosenbloom, P. S.; and Newell, A. 1986. Chunking in Soar: The anatomy of a general learning mechanism. *Machine Learning* 1:11–46.

Langley, P., and Choi, D. 2006. A unified cognitive architecture for physical agents. In *Preceedings of the Twenty-First National Conference on Artificial Intelligence*. Boston: AAAI Press.

Li, N.; Stracuzzi, D. J.; Langley, P.; and Nejati, N. 2009. Learning hierarchical skills from problem solutions using means-ends analysis. In *Proceedings of the 31st Annual Meeting of the Cognitive Science Society*. Amsterdam, Netherlands: Cognitive Science Society, Inc.

Nason, S., and Laird, J. E. 2005. Soar-RL: Integrating reinforcement learning with Soar. *Cognitive Systems Research* 6(1):51–59.

Newell, A. 1990. *Unified Theories of Cognition*. Cambridge, MA: Harvard University Press.

Orkin, J. 2006. Three states and a plan: The AI of F.E.A.R. In *Game Developers Conference*.

Stracuzzi, D. J.; Li, N.; Cleveland, G.; and Langley, P. 2009. Representing and reasoning over time in a symbolic cognitive architecture. In *Proceedings of the 31st Annual Meeting of the Cognitive Science Society*. Amsterdam, Netherlands: Cognitive Science Society, Inc.