Towards Cognition-level Goal Reasoning for Playing Real-Time Strategy Games

Héctor Muñoz-Avila HEM4@LEHIGH.EDU
Dustin Dannenhauer DTD212@LEHIGH.EDU
Computer Science and Engineering, Lehigh University, Bethlehem, PA 18015 USA

Michael T. Cox MICHAEL.COX@WRIGHT.EDU
Wright State Research Institute, Wright State University, Dayton, OH 45435 USA

Abstract

We describe a 3-layer architecture for an automated Real-Time Strategy game player. In these kinds of games, opponent players create and manage large armies of units to defeat one another requiring strategic thinking. Players give commands asynchronously, requiring rapid thinking and reaction. Our 3-layer architecture builds on current automated players for these kinds of games, which focuses on rapid control. The first layer is the control layer that implements the standard reactive player in RTS games. The second layer is a goal reasoning mechanism; it selects goals that are executed by the control layer. The second layer reasons at the cognitive level; it introduces symbolic notions of goal and examines the outcomes of its own decisions. The third layer introduces a meta-reasoning layer that reasons strategically on long-term plans. Our ideas are grounded by using the MIDCA cognitive architecture.

1. Introduction

Like chess, real-time strategy (RTS) games are a form of simulated warfare, where players must maneuver their "pieces" (called "units" in the RTS games parlance) to defeat an opponent. However, RTS games are much more complex than chess due to the following 4 factors: (1) the size of the search space; (2) the partial observability of the state; (3) an infinite number of initial game configurations; and (4) the asynchronous nature of gameplay. We will expand later on these points but the increase in complexity can be illustrated by the fact that, whereas the best automated chess player defeated the best human chess player 10 years ago, top human RTS games players easily defeat the best automated players. Indeed for the RTS game StarCraft,[1] the winner of the AIIDE-13 automated player competition was defeated by a 50th ranked player in the world in about 10 minutes in games where equally skilled human players could take 30 minutes or longer as witnessed during that competition by the authors. This, in spite of the fact that the automated player could issue moves at a rate about 3 times faster than the human player (the exact measure is called APM – or actions per minute).

[1] http://en.wikipedia.org/wiki/StarCraft

The combination of the 4 factors above suggests that we will not see automated players that play at the human level by brute force any time soon. The fact that expert humans defeat so easily expert automated players despite having a much lower APM measurement also suggests the need for automated players that can reason at a high-level of granularity; not only at the object level (i.e., reasoning about individual unit's moves) but also more abstractly (e.g., reasoning about the strategic notion of "defense").

Motivated by these observations, we explore the idea of using cognitive architectures that we hypothesize will result in more advanced automated players. We are particularly interested in examining how these architectures can be used to create goal reasoning mechanisms of varied levels of granularity. We will examine general properties of cognitive architectures and see how they match the needs for effective automated players in RTS games. We also examine challenges of using cognitive architectures for this purpose. Finally, we provide an example highlighting the potential use of the MIDCA cognitive architecture to create an automated player.

2. Real-Time Strategy Games

Real-time strategy game players perform 4 kinds of actions: harvest, construct, build, and destroy. The player needs to harvest resources by using specialized units to collect these resources. These resources can be used to construct structures such as barracks, factories and defensive towers (these structures attack enemy units within their range). Structures such as barracks and factories are needed to build units such as foot soldiers (assembled in barracks) and tanks (built in factories). Building these units also consume resources. Units are used to attack the opponent during combat (when enemy units and/or buildings are within range of friendly units or buildings). Combat follows a paper-rock-scissor model where some class of units are strong against units of another class but weak themselves against a third class. This encourages using a variety of tactics since no class of units is stronger than all others. Furthermore, as a rule, units that are relatively stronger than other units consume more resources when built. Hence, players must reason for particular situations if it will be more cost effective to build more of the weaker units or fewer of the strong units.

As mentioned in the introduction, there are four elements that makes RTS games more complex than chess. We now analyze these challenges in some level of detail.

2.1 Size of Search Space

In Chess, players compete in an 8x8 grid with each player controlling 32 pieces. In contrast, players can control around 100 units (each can be one of dozens of classes), dozens of structures (again multiple classes), and the game takes place in 100x100 grids in RTS games such as StarCraft. More recent games such as Supreme Commander, allow players to control hundreds of units on 1000x1000 grids. Furthermore, each cell in the grid can be of different types including mountain (an impassable obstacle by land units), water (only passable by naval units), and mineral resources (which can be harvested). A careful analysis shows that the game tree for turn-based versions of RTS games is several orders of magnitude larger than chess (Aha, Molineaux & Ponsen, 2005). Briefly, whereas in an average game of chess a player can make around 80 moves, in a strategy game players can make several hundred. Furthermore, the average branching factor of the game tree for chess is 27, whereas for a strategy game, it is on the order of 200. The latter

can be illustrated by the fact that a strategy player at any point must decide where to move its units, whether to construct buildings and where to place them, whether to spend resources on research to upgrade units and if so, choose which kind of research, among other possible decisions.

2.2 Starting Game Configurations

Games are played on a grid or map that may have a number of geographical features including mountains (impassable for land units), rivers, lakes and oceans (also impassable for land units), and resources that players can harvest. If the size of the map is fixed, say 1000x1000 cells, and there are 2 resources then the number of map configurations is $5^{1000x1000}$. If the size of the map varies and is unbounded, then the number of maps is infinite. In addition, each player can start with a base (a collection of buildings placed contiguous to one another) and some units. Typically, the starting base consists of a single building (a town center, which can produce worker units) and a worker (which can harvest resources or build structures).

2.3 Partial State Observability

In RTS games, players only see the area that is in visual range of their units and buildings (typically a few cells around the unit/building). When the game starts, this means that the player only see its base and surrounding areas. As the player moves its units, it will uncover the map configuration. However, opponent's units will remain unseen unless they are in visual range of the player's own units. This is referred to as the fog of war since the opponent's movements might be hidden.

2.4 Asynchronous Nature of Gameplay

In RTS games, players make their moves (i.e., commands) asynchronously. These commands include directing a unit to move to a cell or ordering a worker to construct a building in a location (typically a group of contiguous cells forming a rectangular shape). This means that the speed to issue the commands is an important success factor. Alleviating this is the fact that the game engine automates some actions. For example, a worker tasked with harvesting resources will continue to do so until it is issued new commands or is killed by an opponent or the resources are exhausted. Importantly, units will attack opponent's units unless the player explicitly has command them not to do so. Nevertheless, in human competitions, particularly in later stages of the game where each player is controlling hundreds of units and dozens of buildings spread out over a map, players can be seen frantically issuing orders.

3. Automated RTS Players at the Object Level

For the purposes of this discussion, we distinguish between reasoning about the environment and the environment itself that constitutes the *ground level*. We further distinguish between reasoning at the *object* (i.e., cognitive) *level* and at the *meta-level* (i.e., metacognition) (Cox & Raja, 2011). The ground level refers to the maps, units, and player's actions. Existing automated players (such as those used in commercial RTS games or those used in the StarCraft automated player competition) all reason at the object level about the ground level. These programs can exhibit complex strategies: some will try to attack the opponent continuously draining the opponent's

resources, while others slowly build a powerful army to attack the opponent at a later stage of the game. Others analyze the map and methodically take control of resources until they control most resources enabling them to overwhelm the opponent by producing large numbers of units that the opponent cannot possibly counter. These systems still reason at the object level; the strategies, while undoubtedly complex, are selected based solely on circumstances of the ground level. The control-resource-strategy for example, will pick the nearest resource to the base that is undefended, and expand in that direction. Hardcoded doesn't mean lack of flexibility; the automated player will change its strategy adapting to previously foreseen circumstances.

4. Goal Selection in RTS Games at the Object Level

A key characteristic of cognitive systems is the capability for high-level cognition and goal reasoning. That is, the capability of selecting goals of multiple levels of abstraction, determining how to achieve those goals with multi-step reasoning mechanisms and introspectively examine the results of those decisions.

Recent efforts on RTS game research have begun to explore cognition at the object level. Specifically, we examine agents that exhibit *goal-driven autonomy (GDA)* (Aha, Klenk, Munoz-Avila, Ram, & Shapiro, 2010; Cox, 2007; Klenk, Molineaux, Aha, 2013; Munoz-Avila, Aha, Jaidee, Klenk, & Molineaux, 2010). GDA agents are those that (1) generate a plan to achieve the current goal; (2) monitor the execution of the plan and detect any discrepancy between the expectations (e.g., a collection of atoms that must be true in the state) and the actual state; (3) explain reasons for this discrepancy; and (4) generate new goals for the agent to achieve based on the explanation generated. The following are three instances of automated GDA agents for RTS games:

- Weber, Mateas, and Jhala (2012) learns GDA knowledge while playing the RTS game StarCraft. The architecture of the automated player uses the idea of tasks managers which are components specialized for tasks such as combat and construction (we will expand on these task managers later on). Most of these task managers are hardcoded but the manager responsible for building units uses GDA. This enables the automated player to dynamically change the production of units to accommodate for changes in the environment. Weber (2012) uses vectors of numbers as its representations. These numbers represent counters for elements in the game such as number of soldiers. So when a discrepancy occurs because, say, the system expected to have 12 soldiers but it only has 8, it will generate a new goal to generate 4 soldiers.
- Jaidee, Munoz-Avila, and Aha (2011) learns and reasons with expectations, goals and how to achieve those goals. It has two main differences in comparison to Weber, Mateas, and Jhala: (1) it neither learns nor reasons with explanations for failure reasons; and (2) GDA is used to control all aspects of the gameplay as opposed to only production of new units. It uses multiple-agents, one for each type of unit or building in the game. The GDA cycle assigns goals for each of these agents to achieve ensuring coordination between the agents.
- Dannenhauer and Munoz-Avila (2013) showcases a GDA agent that takes advantage of ontological information in games. This enables the system to reason with notions such as

controlling a region. A rule can define the concept by indicating that a region is controlled by agent A, if at least one unit of agent A is in the region and no unit of agent B is present. The GDA cycle uses these notions as part of its reasoning with expectations process.

Each of these three agent variations have been shown to perform well against hard-coded opponents in a variety of experimental settings.[2]

5. High-Level Goal Reasoning

GDA agents exhibit some of the characteristics of cognitive systems (Langley 2012). In particular, they use a structured representation of knowledge:

- **Symbolic structures**. GDA systems frequently use the notion of goals, states and plans based on the STRIPS formalism, as such these symbols are interpretable symbolic mechanisms. In the context of games, goals refer to conditions in the state that must be held true. For example, controlling a resource in a specific location (e.g., having two or more units and a building around the resource). As such goals refer to conditions at the object level.
- **Complex relations.** For example, a plan is a sequence of interrelated actions and their sequencing is dependent on cause-effect relations between the tasks. For instance, a plan to control a resource might call to produce a mixture of units and when these units are produced, send them to the location of the resource.

Another characteristic of cognitive systems, GDA systems perform heuristic search; they cannot guarantee optimal solutions in these kinds of highly dynamic environments and very large decision spaces. While mini-max algorithms, which guarantee some form of optimal behavior to counter an opponent's move, have demonstrated to be useful to RTS games (Churchill, Saffidine, & Buro, 2012), they focus on small combat encounters involving a handful of units. Similarly, learning algorithms have been used to determine best opening moves in the game that maximize economic output within the first few minutes of the game but these involve only early construction of buildings and worker resource harvesting tasks; they are based on the premise that (1) at early stages in the game there will be no combat and (2) gameplay decisions are localized around the starting base. Cognitive systems introspectively examine their own decisions based on the interactions with the environment and their own knowledge to tune its own decision making.

High-level cognition enables abstract reasoning that goes beyond reasoning at the object level. This is a significant departure from existing RTS automated players. We will ground our ideas providing discussions by basing the agent on the MIDCA cognitive architecture.

6. The MIDCA Cognitive Architecture

Computational metacognition distinguishes reasoning about the world from reasoning about reasoning (Cox, 2005). As shown in Figure 1, *the Metacognitive, Integrated, Dual-Cycle*

[2] They have not been tried in competition settings because competition rules would need to be refined to account for the fact that these agents learn their knowledge.

Architecture (MIDCA) (Cox, Oates, & Perlis, 2011; Cox, Oates, Paisner, & Perlis, 2012) consisting of "action-perception" cycles at both the cognitive (i.e., object) level in orange and the metacognitive (i.e., meta) level in blue. The output side of each cycle consists of intention, planning, and action execution; the input side consists of perception, interpretation, and goal evaluation. At each cycle, a goal is selected and the agent commits to achieving it. The agent then creates a plan to achieve the goal and subsequently executes the planned actions to make the domain match the goal state.[3] The agent perceives changes to the environment resulting from the actions, interprets the percepts with respect to the plan, and evaluates the interpretation with respect to the goal. At the object level, the cycle achieves goals that change the environment or

[3] Note that this does not preclude interleaved planning and execution. The plan need not be fully formed before action execution takes place.

ground level. At the meta-level, the cycle achieves goals that change the object level. That is, the metacognitive perception components introspectively monitor the processes and mental-state changes at the cognitive level. The action component consists of a meta-level controller that mediates reasoning over an abstract representation of the object-level cognition.

Furthermore, and unlike most cognitive theories, our treatment of goals is dynamic. That is, goals may change over time; goals are malleable and are subject to transformation and abandonment (Cox & Zhang, 2007; Cox & Veloso, 1998). Figure 1 shows *goal change* at both the object and meta-levels as the reflexive loops from goals to themselves. Goals also arise from

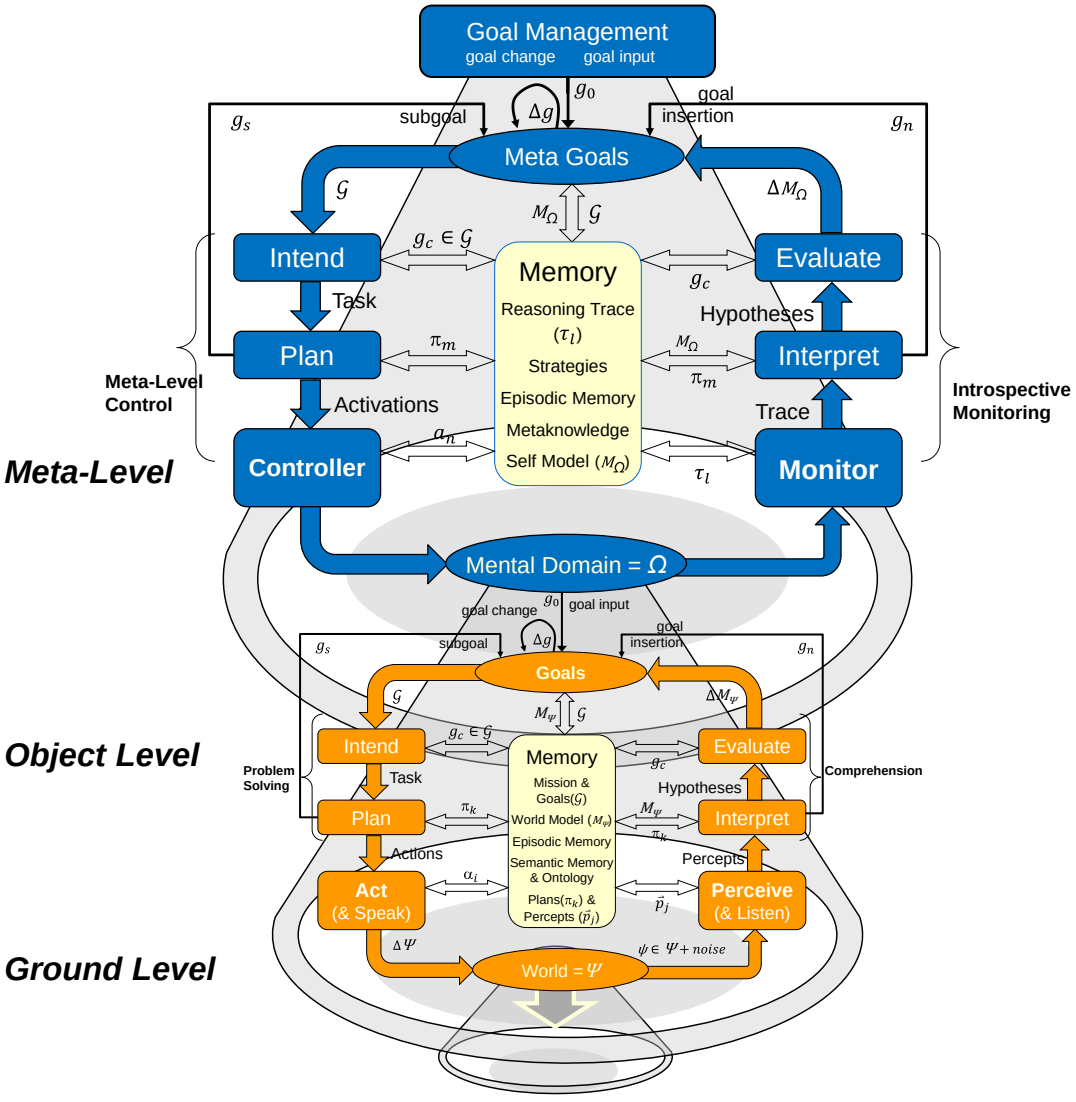

Figure 1. Metacognitive, Integrated, Dual-Cycle Architecture (MIDCA)

traditional *sub-goaling* on unsatisfied preconditions during planning (the thin black back-pointing arrows on the left of both blue and orange cycles). Finally new goals arise as MIDCA detects discrepancies in the input given its expectations. It explains what causes the discrepancy, and generates a new goal to remove the cause (Cox, 2007). This type of operation, called *goal insertion*, is indicated by the thin, black arrows from the interpretation processes in the figure.

Goal insertion is the fundamental GDA process in MIDCA and occurs at both the object and meta-levels. At the object level, perception provides observations, and plans from memory to provide the expectations. The interpretation process detects discrepancies when observations conflict with expectations and will then explain what caused the discrepancy and will generate a new goal. At the meta-level, monitoring provides an observation of a trace of processing at the object level and a self-model provides the expectations. Like the object-level GDA process, interpretation produces an explanation of why the reasoning at the object level failed and uses the explanation to generate a learning goal to change the knowledge or reasoning parameters of the object level. An explanation in MIDCA is a causal knowledge structure, χ, consisting of a set of antecedents that together cause the discrepancy.[4]

$$\chi = \delta \rightarrow \ldots \rightarrow s_c$$

According to Cox (2007; this volume), the explanation contains a *salient antecedent*, δ, that represents the root cause of the problem signaled by the discrepancy. The goal, $g_c = \neg \delta$, is then to achieve the negation of this antecedent, therefore removing the discrepancy and solving the problem.

Numerous ways exist that the meta-level can affect the object level. The meta-level can act as an executive function in a manner similar to CLARION (Sun, Zhang, & Mathews, 2006). It can decide between object-level parameters, it can allocate resources between competing object-level processes, and it can set priorities on object level goals, swap object-level processes (i.e., change the planner with a different one) and it can also insert goals at the object level.

7. MIDCA in RTS Games

The premise of our idea to apply MIDCA in RTS games is for the lower level cycle to act on the RTS action (ground) level and the higher level on the RTS reasoning (object) level. That is, the MIDCA cycle at the object level, monitors the situation at the ground level and generates goals and the plans to achieve those goals. Since planning in RTS games is a complex activity within itself we will borrow the idea of *managerial tasks* (Scott, 2002), used by many automated RTS games, to generate and execute these plans. Managerial tasks are performed by the following components:

1. **Building**. In charge of structure building, including keeping track of the order in which buildings must be constructed (e.g., a factory requires barracks).

[4] The explanation is actually a graph $\chi = (V, E)$ with $\delta \in V$ an element of the source nodes and $s_c \in V$ a distinguished element of the sink nodes. See Cox (2011) for further details.

2. **Units**. Responsible of creating new units and prioritizing the order in which units are created.
3. **Research**. Responsible for creating new technology that enables the creation of more powerful units.
4. **Resource**. Responsible for gathering resources (i.e., minerals and vespene gas), which is needed to construct units, buildings and invest on research.
5. **Combat**. Responsible for controlling units during combat; determining which units to attack, or whether to defend.
6. **Civilization**. Responsible for coordinating the managers.

Each of these components could be a learning component although for a first implementation we are going to use hard-coded implementations of these. This will enable us to directly use existing automated players such as Skynet or UAlbertaBot and have the MIDCA architecture build on top. This is how we implemented our GDA agent playing StarCraft (Dannenhauer & Munoz-Avila, 2013). But unlike that agent, MIDCA will enable meta-reasoning.

For the higher-level cycle, MIDCA will monitor the decisions made at MIDCA's object level and intervene (by changing or assigning new goals, or by adjusting parameters in object level components such as the planner similar to Dannenhauer, Cox, Gupta, Paisner & Perlis, 2014). The higher level acts as a long-term, "broad perspective" reasoning mechanism. It reasons not only on information about the object level but also on the knowledge used by the object-level MIDCA to generate its goals. We believe this is a crucial reasoning capability, one that is missing in existing automated players.

Our automated player performs asynchronous decision-making at three levels. We describe them from the most concrete to the most abstract:

- **Object-level reactive control**. The player decisions are made by the six managers described above. This ensures immediate and continuous control of all units and buildings. This is the level programmed by most commercial RTS players and entries to the automated player tournaments. It controls everything from the production of units, harvesting of resources and combat. The civilization manager ensures that a default strategy, such as the systematic control of resources in the map, is pursued.
- **Object-level goal formulation**. This is performed by the object level from MIDCA and is reminiscent of GDA automated players in the sense that it monitors the current state of the game and triggers new goals and the means to achieve them (i.e., plans). But, unlike existing GDA players, it has a symbolic model of the goals achieved by the object-level reactive controller and their outcomes. This enables the goal formulation component to monitor the execution and formulate new goals. For example, if the opponent launches an assault on a base built around resources, by default, the object-level reactive control might direct nearby units to defend the base. The object level goal formulator might detect that the opponent is gaining the upper hand and will take control of the resource. The goal formulator might generate a new goal: to re-take the resource or to, instead, take over an opponent's controlled

resource that is less defended. This goal will supplement the default strategy of the object-level reactive control; tasking, with higher priority, the managers to take actions towards achieving the new goal. This ensures consistent behavior where these goals are given priority but other goals, either from the default strategy or previously stated goals are still being pursued (but with less priority).

- **Meta-level long-term control**. This is performed by the MIDCA's metacognitive level. It monitors the decisions of the object level goal formulation and the object level reactive control and might override decisions made and tasks new goals to achieve. For example, the cognitive level might invalidate the goal to re-take or to take an alternative resource because it determines that the reactive control is about to take the enemy starting base and therefore end the game in victory. In this case, it will invalidate the new goal as accomplishing it will consume resources that might detract from the imminent victory. The meta-cognitive level reasons at a higher level of granularity reasoning on the lower level's own reasoning and considering long-term implications.

Because of the real-time nature of the games, these modules operate in parallel to the MIDCA level. This guarantees that the automated player, MIDCA-RTS, will always react to immediate changes in the situation (e.g., a sudden attack on our base) while the metacognition level reasons on strategic decisions that go beyond immediate game occurrences.

8. Example Scenarios

8.1 Detecting a Feign Attack

A common situation that occurs in RTS games is for the enemy to harass your harvesting units, often with a single worker unit of their own. This behavior has also occurred during automated player matches (Skynet is known for using a worker to harass enemy workers very early in the game). In this scenario, there are two common approaches that are problematic for the defending player. First, the player could ignore the enemy probe, but this will cause workers to be killed. The second is to send all worker units to attack the probe. This is problematic because the probe will lead the workers on a chase (as seen in Figure 2) and resource harvesting will be nearly stopped. The orange object level will be able to respond to the discrepancy of the enemy probe, but the meta-cognitive blue layer will figure out / learn which approach is the best, which is usually to send 1 of our workers to attack the enemy worker, and let the rest keep harvesting.

Figure 2. The workers following an enemy's probe

8.2 Performing a Feign Attack

Using the concept of distract, MIDCA could perform strategies at a higher level, such as distracting the enemy with a small force of units at the front of their base and concurrently sending units via dropships behind the enemy base. In Figure 3, air ships have entered the back of the base from the left side and have

dropped units to destroy buildings (in this image some buildings have already been destroyed and two are on fire). This is happening while the base's defenses are busy with the 'distract' group at the front of the base (not seen in the picture). This example motivates the benefit of higher level concepts such as distract, used by a cognitive architecture such as MIDCA.

Figure 3. Assault from behind the enemy's base

8.3 Performing a Feign Attack

In RTS matches, there are often crucial points on the map that are strategically important. The concept of 'defense' is another high-level concept that a cognitive architecture such as MIDCA could infer. While some automated bots already exhibit this behavior in specific situations, it is not an explicit concept. The agent will be able to carry out more sophisticated attacks if the notion of defense is explicit and flexible, because one could imagine not only defending a chokepoint (see Figure 4) or map location, but also a particularly strong offensive unit vulnerable to specific enemies. For example, it is common to pair a siege tank with melee units to protect the tank from enemies that get close enough to damage the tank without being fired upon.

Figure 4. Feign attack to front of the base

9. Final Remarks

While attaining automated players for RTS games that can play at the human level are an interesting challenge in their own right, they also can be seen as a challenge for agents that can exhibit high-level cognition. The latter is one of our main motivations with this research. We believe that the three layer architecture proposed will guarantee the reactive behavior needed for these kinds of games, while the meta-cognition will enable new high-level capabilities currently not exhibited in existing automated players.

Acknowledgements

This work is funded in part under NSF grant 1217888. Further support has been provided by ARO under grant W911NF-12-1-0471 and by ONR under grants N00014-15-1-2080, N00014-12-1-0430, and N00014-12-1-0172. We thank the anonymous reviewers for their insights and comments.

References

Aha, D. W., Klenk, M., Munoz-Avila, H., Ram, A., & Shapiro, D. (Eds.) (2010). *Goal-driven autonomy: Notes from the AAAI workshop*. Menlo Park, CA: AAAI Press.

Aha, D.W., Molineaux, M., & Ponsen, M. (2005). Learning to win: Case-based plan selection in a real-time strategy game. *Proceedings of the Sixth International Conference on Case-Based Reasoning* (pp. 5-20). Chicago, IL: Springer.

Cox, M. T. (this volume). Toward a formal model of planning, action, and interpretation with goal reasoning. To appear in *Proceedings of the 2015 Annual Conference on Advances in Cognitive Systems: Workshop on Goal Reasoning* (IRIM Tech Report). Atlanta: Georgia Institute of Technology.

Cox, M. T. (2011). Metareasoning, monitoring, and self-explanation. In M. T. Cox & A. Raja (Eds.) *Metareasoning: Thinking about thinking* (pp. 131-149). Cambridge, MA: MIT Press.

Cox, M. T. (2007). Perpetual self-aware cognitive agents. *AI Magazine 28*(1), 32-45.

Cox, M. T. (2005). Metacognition in computation: A selected research review. *Artificial Intelligence*. *169* (2), 104-141.

Cox, M. T., Oates, T., Paisner, M., & Perlis, D. (2012). Noting anomalies in streams of symbolic predicates using A-distance. *Advances in Cognitive Systems 2*, 167-184.

Cox, M. T., Oates, T., & Perlis, D. (2011). Toward an integrated metacognitive architecture. In P. Langley (Ed.), *Advances in Cognitive Systems: Papers from the 2011 AAAI Fall Symposium* (pp. 74-81). Technical Report FS-11-01. Menlo Park, CA: AAAI Press.

Cox, M. T., & Raja, A. (2011). Metareasoning: An introduction. In M. T. Cox & A. Raja (Eds.) *Metareasoning: Thinking about thinking* (pp. 3-14). Cambridge, MA: MIT Press.

Cox, M. T., & Veloso, M. M. (1998). Goal transformations in continuous planning. In M. desJardins (Ed.), *Proceedings of the 1998 AAAI Fall Symposium on Distributed Continual Planning* (pp. 23-30). Menlo Park, CA: AAAI Press.

Cox, M. T., & Zhang, C. (2007). Mixed-initiative goal manipulation. *AI Magazine 28*(2), 62-73.

Churchill, D., Saffidine, A., & Buro, M. (2012). Fast Heuristic Search for RTS Game Combat Scenarios. In *AIIDE*.

Dannenhauer, D. and Munoz-Avila, H. (2013) LUIGi: A Goal-Driven Autonomy Agent Reasoning with Ontologies. *Advances in Cognitive Systems Conference (ACS-13)*.

Dannenhauer, D., Cox, M. T., Gupta, S., Paisner, M. & Perlis, D. (2014). Toward meta-level control of autonomous agents. In *Proceedings of the 2014 Annual International Conference on Biologically Inspired Cognitive Architectures: Fifth annual meeting of the BICA Society* (pp. 121 -126). Elsevier/Procedia Computer Science. Forbus, K. D. (1984). Qualitative process theory. *Artificial Intelligence*, *24*, 85–168.

Jaidee, U., Munoz-Avila, H., Aha, D.W. (2011) Integrated Learning for Goal-Driven Autonomy. To appear in: Proceedings of the Twenty-Second International Conference on Artificial Intelligence (IJCAI-11). AAAI Press.

Klenk, M., Molineaux, M., & Aha, D. (2013). Goal-driven autonomy for responding to unexpected events in strategy simulations. *Computational Intelligence*, *29*(2), 187–206.

Munoz-Avila, H., Aha, D.W., Jaidee, U., Klenk, M., & Molineaux, M. (2010). Applying goal driven autonomy to a team shooter game. *Proceedings of the Twenty-Third Florida Artificial Intelligence Research Society Conference* (pp. 465-470). Menlo Park, CA: AAAI Press.

Langley, P. (2012). The cognitive systems paradigm. *Advances in Cognitive Systems 1*, 3–13.

Scott, B. (2002) Architecting an RTS AI. AI game Programming Wisdom. Charles River Media.

Sun, R., Zhang, X., & Mathews, R. (2006). Modeling meta-cognition in a cognitive architecture. *Cognitive Systems Research, 7*, 327-338.

Weber, B., Mateas, M., & Jhala, A. (2012). Learning from Demonstration for Goal-Driven Autonomy. In *AAAI*.