A Micromanagement Task Allocation System for Real-Time Strategy Games

Keith D. Rogers and Andrew A. Skabar

***Abstract*—Real-time strategy (RTS) game play is a combination of strategy and micromanagement. While strategy is clearly important, the success of a strategy can depend greatly on effective micromanagement. Recent years have seen an increase in work focusing on micromanagement in RTS AI, but the great majority of these works have focused on policies for individual units or very specific situations, while very little work has aimed to address the need for a broadly applicable structure for unit group coordination. This paper conceptualizes RTS group level micromanagement as a multiagent task allocation (TA) problem, and proposes the micromanagement task allocation system (MTAS) as a framework to bridge the gap between strategy and unit level micromanagement. MTAS fits into the common layered RTS AI structure, and can thus, in principle, be integrated into any existing system based on this layered structure. We describe the integration of MTAS into E323AI (an existing AI player for the spring RTS engine), and provide empirical results demonstrating that MTAS leads to statistically significant improvement in performance on the popular RTS game *Balanced Annihilation*.**

***Index Terms*—Artificial intelligence (AI), computer games, micromanagement, real-time strategy (RTS), task allocation (TA) systems.**

I. Introduction

REAL-TIME STRATEGY (RTS) games are a popular computer game genre that consists mostly of war games that require the player to collect resources, construct and defend bases, and amass an army or build a nation that eventually overwhelms all opponents. As computers become ever more powerful, computer games in many genres continue to utilize this technology to create more intricate and absorbing experiences. Increases in computing power have allowed the RTS genre to create environments with complex and interactive terrain that can contain many hundreds and even thousands of objects with many possible interactions. This same increase in computing power is also making it possible to explore ever more elaborate and complex AI [1]. Research into AI systems that provide challenging opponents, cooperative allies, and entertaining dynamic environments is increasingly attracting attention from both academia and industry [2].

Manuscript received February 07, 2013; revised July 08, 2013 and October 20, 2013; accepted December 15, 2013. Date of publication January 02, 2014; date of current version March 13, 2014.

The authors are with the Department of Computer Science and Computer Engineering, La Trobe University, Melbourne, Vic. 3086, Australia (e-mail: krogers@students.latrobe.edu.au; a.skabar@latrobe.edu.au).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2013.2297334

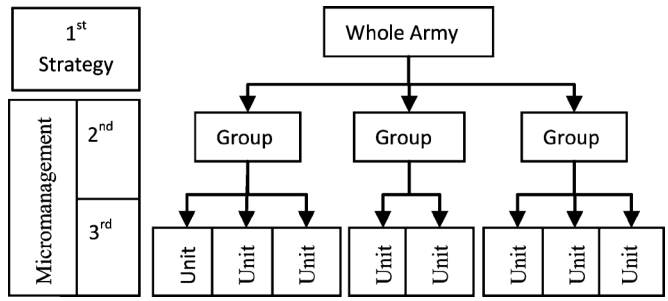

Fig. 1. Typical three-layer RTS AI structure.

RTS game play is often viewed as a combination of strategy and micromanagement. Strategy relates to high-level decision making, such as where to build a base, which types of units and technologies to focus on, and when to attack and defend. In contrast, micromanagement is concerned with the implementation of the strategy, coordinating the actions of individuals or small groups of units in an effort to achieve the high level goals.

Because of this separation between strategy and micromanagement, most AI designers use a layered approach in which lower levels of the AI provide progressively higher levels of abstraction. This separation allows higher layers to focus on general strategies, while lower levels handle the details of how these strategies are executed, similarly to a military chain of command [3]. For example, Fig. 1 shows a typical breakdown into three layers. The top layer is responsible for strategic planning and distributing goals to groups. Groups are responsible for taking a more detailed look at their local situation, and carrying out these goals by giving orders to individual units. Units are then responsible for carrying out their individual orders to the best of their ability. By breaking down the AI into three layers and identifying the critical requirement of each layer, AI techniques that best suit each layer can then be used [3].

The focus to date in RTS AI has been on improving strategy and strategic-level planning. Techniques such as case-based reasoning [2], [4]–[9], player modeling with Bayesian networks [10], [11] and neural networks [12], build order optimization using tree search heuristics [13], and reinforcement learning through dynamic scripting [14] have all been successfully applied to devising good strategies. However, devising a good strategy, although very important and very challenging, is only halfway to being a good RTS player. When players have hundreds of objects under their control, the increased unit performance gained through effective micromanagement can be the deciding factor, particularly in situations where groups of units engage in combat [15]. Any strategy, brilliant though it may be, can be undermined by poor micromanagement [16].

Despite the importance of micromanagement, it has only recently become a focus of RTS AI research. Although there has been some success in improving a subset of micromanagement situations, there has been little success in finding a structure that facilitates a holistic approach to micromanagement that can be applied to a wide variety of unit types and situations.

The main contribution of this paper is a system which we refer to as the micromanagement task allocation system (MTAS). MTAS is a flexible AI framework applicable to a wide variety of unit types and micromanagement situations in a range of RTS games, and provides effective unit group micromanagement in pursuit of goals devised by the strategic level AI. Thus, while MTAS is responsible for coordinating the actions of a collection of units of any unit type and number, it is not responsible for making strategic decisions such as moving the group's location, retreating in the face of impossible odds, or moving to reinforce other attacking or defending groups. MTAS has been designed to fit into the common layered RTS AI structure described above, and thus should, in principle, be able to be integrated into any existing system that is based on this layered structure. We report on the successful retrofitting of MTAS into E323AI, a preexisting AI player for the spring RTS engine's *Balanced Annihilation* game, and present empirical results that demonstrate that the incorporation of MTAS leads to a statistically significant improvement in terms of game wins.

The paper is structured as follows. Section II provides an overview of current work in the field of micromanagement. Section III then examines RTS micromanagement as a task allocation (TA) problem, and identifies the characteristics which differentiate it from more conventional TA problems. Section IV presents the proposed micromanagement system, and describes its integration with the strategic-level AI component, and Section V details the specific implementation of MTAS into E323AI. Section VI provides empirical results and a discussion of the system's success in the popular RTS game *Balanced Annihilation*. Finally, Section VII draws conclusions, outlines plans for future work, and comments on the potential for MTAS to be incorporated into other currently isolated RTS AI research projects.

II. Related Work

Early approaches to micromanagement in RTS games were based on predefined static scripts for each type of group behavior (i.e., attacking, defending, base building, etc.) [4], [17]–[20]. However, scripted micromanagement suffers from the same issues as scripted strategies: poor performance in unforeseen situations, predictability, exploitable weaknesses, little or no portability, and the need for a lot of effort and expert knowledge to implement. In the absence of effective micromanagement, strategy-level reasoning components were limited in the selection of viable options, as they were forced to favor plans that do not rely heavily on micromanagement [19]. Fortunately, the last few years have seen an increase in attention to RTS micromanagement, no doubt motivated by the increasing popularity of RTS AI competitions such as the annual Starcraft AI competitions run at the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment.

A significant issue in RTS micromanagement is the multibody spatial reasoning involved in positioning the many units under the AI's control in order to reduce collisions, attack enemy units, and avoid enemy attacks. A number of different approaches have been used to address this issue. Potential fields have been applied to unit group movement coordination [21], risk avoidance path planning [22], and tactical unit positioning [23]. Jurney [24] presents a rule base and virtual squad leader combination that provides more natural looking coordinated squad movement and tactical positioning for squads in the commercial RTS game *Company of Heroes*.

Recent work has also addressed the extended problem of coordinating unit movement while either attacking or under attack. Neural networks have displayed improvements in deciding which individual military units should attack and/or move [25], and reinforcement learning [26] and evolutionary computing [27] have been shown to improve the more specific fight or flight decision on an individual unit level. The winners of the 2010 AIIDE annual Starcraft AI competition (Berkeley's Overmind) used dynamic potential fields to coordinate a "hit and run" tactic that proved very effective when tuned for squads of Mutalisks (flying units that do not suffer from the complications of unit collisions, terrain collisions, momentum, or turning circles).

The Bayesian model of unit move selection presented by Synnaeve and Bessiere [28] not only makes the decision of whether to move or attack, but also which direction to move, and attempts to minimize unit collision that would hinder allied units. A target selection heuristic and finite state machine are used to assign goals to the individual units, who then use a Bayesian inference network that incorporates influence maps, a prebuilt network ontology (based on expert knowledge), and learned weights to decide which direction to move.

Churchill *et al.* [29] investigated the feasibility of a fast heuristic tree search model. In their system, RTS micromanagement is viewed as a very fast simultaneous move turn-based game (a valid approach if you consider that each AI player is actually updated in discrete intervals, typically every 30th of a second). Churchill *et al.* [29] present a system that is able to accommodate "moves" that span multiple turns (an issue unhandled by traditional heuristic tree search models). This approach differs from other current work because it coordinates the actions of an entire group of units (not just one unit) by searching though possible moves and counter moves for all allied and enemy units in the scenario to try and allocate a sequence of actions to units in the group that leads to the most favorable outcome. The system proved viable within the constraints of a single battle consisting of two equal teams of up to eight units, where units can only move and attack with a small variety of attack strengths, ranges, and movement speeds. However, in situations where there are larger battles on multiple fronts, with units that can use many different attack methods, spells, and aura effects, and must deal with the complications of terrain, fog of war, imprecise damage/accuracy of attacks, unequal opposing unit groups, the arrival of reinforcements (enemy and/or allied), and many different possible objectives, the explosion in computational complexity would most likely render this style of reasoning about action selection too slow.

With the exception of [29] current work in the area of micromanagement is focused on unit control on an individual basis (though some systems create the illusion of cooperation). The micromanagement systems in AIIDE Starcraft competition AIs, such as Berkeley's Overmind, still lack generality as they are tailored for a particular unit type and/or situation, and the strategy component plays to support that unique strength [30]. While other works exploring neural networks, potential fields, Bayesian inferencing, and other architectures and learning algorithms display improvements in their targeted aspect or situation of micromanagement, they have at present only been assessed in simple scenarios that isolate the aspect of micromanagement in question, often with a very restricted number of units in each battle and/or using very few unit types. Nevertheless, they are a clear demonstration that there are gains to be had through effective micromanagement.

While there is no doubt that there is progress being made in RTS micromanagement, there is a lack of work into providing a general structure capable of bridging the gap from unit to strategy; in other words, a system for micromanaging groups of units that can utilize all these unit level techniques, scale well to a tolerable number of units and unit types, be given a wide selection of goals, report back to the strategy upon completion/failure/invalidation, and be generally applicable to a range of RTS games and AI players.

III. Micromanagement as a Task Allocation Problem

A significant challenge facing those who wish to compare or improve RTS micromanagement is that there is no clear definition of the problem. For this reason, this section is dedicated to describing our definition of RTS group level micromanagement as a TA problem.

The problem of TA in large- and small-scale distributed systems is an ongoing challenge that has attracted research for many decades, and has applications in a wide variety of problem areas including manufacturing, transport, robotics, and distributed computing [31]. Micromanagement in RTS games can also be viewed as a TA problem, since at any point in time the many different actions that can be assigned to objects in the game can be viewed as tasks. Based on an object's abilities, these tasks may include attacking enemy units, healing friendly units, constructing buildings, moving to a new location, constructing new units, etc. For example, the option of "attacking enemy unit x" becomes an "attack task," with the target being "unit x."

The challenge is to devise a system that distributes these tasks among the available agents (units and buildings) in such a way that the group achieves the goals passed to it from the strategic-level AI. To illustrate how micromanagement in RTS games compares to other TA problems, and to understand the specific challenges of allocating tasks to a group of agents in an RTS environment, this section examines the important conditions that we feel such a TA system must satisfy.

A. Dynamic Task Allocation/Re-Allocation

A major aspect of RTS micromanagement that sets it apart from most other TA problems is the need for dynamic, as opposed to static, allocation/re-allocation of tasks. The adversarial RTS environment and the large number of dynamic attributes that affect which agent would be best for each task means there is no room for blind adherence to the initial task assignment, as it will quickly be rendered suboptimal. These dynamic attributes of tasks and agents change on an individual basis over the course of the game, and may change significantly even within the time taken to complete a task. Thus, the AI must continuously react to the changing situation, reassigning agents to the most appropriate task at that time.

Real-time task re-allocation is only just starting to be explored in the TA literature, and mainly in the context of robotics (see [32]). For example, a common solution explored in the adversarial robotics literature is to search a tree of all possible task distributions in order to find the one with the highest net utility. However, the degree of re-allocation in robotics domains is not anywhere close to the degree required to keep pace in an RTS game. The complexity of RTS micromanagement and the need for constant reevaluation of task assignments would make the tree very large, and trimming and traversing it to find a solution too slow.

The flexible job shop problem (FJSP) literature also contains some work on real-time TA, but because these problems are planning projects that typically span several days, "real time" here refers to a system that can devise a solution within a few minutes, which is less than acceptable in an RTS AI. While there is some work investigating improvements in TA in dynamic environments with dynamic creation and destruction of tasks and agents [33], this work is still yet to include the additional complication of agents and tasks possessing properties that change at runtime, thus affecting the optimality of the TA.

B. Centralized Processing

The requirement for centralized processing comes about because in any game the simulation of the game is maintained by one computer. This renders two significant challenges in multirobot task allocation (MRTA) systems and multiagent systems (MASs) far less relevant: reducing communications, and distributing the work load associated with TA across many processors [31]. Given that TAs need to be reconsidered as the situation evolves, constantly rerunning any of these decentralized negotiation protocols would be cumbersome and inefficient. This shifts the focus away from distribution and communication costs, and toward algorithms that can be interrupted and resumed easily due to the need to maintain an acceptable frame rate in the game.

C. Agent-Oriented Task Allocation

In most TA problems, the number of agents usually equals or exceeds the number of tasks, and there is usually a one-to-one relationship between tasks and agents (i.e., there is only one vacant slot per task to be filled by an agent, and each agent can only pursue one task at a time). This means that the number of vacancies (i.e., the sum of all available slots for each task) is less than or equal to the number of agents. In these environments, the problem is viewed as trying to find the best agent for the task.

This sort of task-oriented system can be seen in MAS protocols, the most prolific of which is Contract Net, a protocol that relies on agents presenting tasks to the other agents in the

system, who then place bids on those tasks, an agent being awarded a contract if it places the best bid for that task [34]. This protocol is successful in systems where there are few simultaneous tasks and many agents, but has been shown to suffer when the number of simultaneous tasks becomes high compared to the number of agents. Agents attempting to juggle many simultaneous multistage negotiations can lead to over-utilization or under-utilization of resources, or drastic increases in the number of communications required.

In a typical RTS game, the situation is reversed because there are often many more tasks than agents, and many of those tasks can be pursued by many agents (i.e., there are many vacancies per task). Therefore, the total number of vacancies will usually far exceed the number of agents. Interestingly, this switches the focus of the problem from trying to find the best agent for each task, to a search for the best task for each agent. We refer to this as agent-oriented TA, which forms an important part of the algorithm presented in Section III.

D. Heterogeneity and Cooperative Task Execution

The variety of different units and buildings and their different abilities makes for a heterogeneous environment. These differences result in each agent being capable of participating in only a subset of all available tasks, and with varying degrees of proficiency for each of those tasks. This is different to many TA problems, in which many agents have exactly the same capabilities and proficiencies. Moreover, in RTS games, it is often desirable for many agents to tackle the one task. Although most tasks can be attempted by one agent, more agents can make it easier/faster to achieve the task. The number of agents that would most efficiently carry out a task depends on the attributes of each contributing agent (a few powerful agents or many weaker agents), and would have to be assessed on a task by task basis. Unfortunately, most of the protocols and heuristics devised for MRTA and FJSP only consider scenarios where single agents are capable of completing single tasks (individual execution).

E. Nondependent Tasks

It does not have to be the case that tasks at the group micromanagement level are defined such that each task can be executed independently of the execution of any other, but we believe this is an acceptable and workable simplification, as demonstrated with the successful implementation of the system described in Section IV. While a high level view of an RTS game has many dependencies between tasks (i.e., to build a soldier you must have a barracks), the group micromanagement level is focused on reacting to the current situation, and doing as best as is possible with the resources assigned by the strategic level AI. The act of attacking enemy targets, building structures, and guarding will often require a sequence of moves in conjunction with the attacking, building, or guarding, but these should be handled by the unit micromanagement so that, from the group's perspective, this series of actions can be viewed as a single task (attack, build, or guard) that is independent of the group level move task (which would coordinate the change in the group's current location).

F. High Throughput

High throughput is the result of the system having to select the best of many options that lead to achieving the goal of delivering the most crippling blow to the enemy, while limiting damage to oneself. If this objective is viewed as completing a sequence of tasks, there are many paths to the goal, and often the more of these tasks that are completed in a given time, the less the damage that will be received.

G. Fault Tolerance

One of the main objectives in an RTS is to destroy all enemy agents. As a result, the importance of fault tolerance is taken to the extreme. Instead of the system needing to handle the occasional agent failure (such as in FJSPs [35]), the system must be able to operate in a war zone where significant degradation and loss of agents is expected.

H. CPU Intensiveness

Over the course of an RTS game, the load on the central processing unit (CPU) varies over time and can become quite high. Unfortunately, this usually coincides with large-scale offensives, which is usually when the AI is presented with the most choices, and decision making is most critical. For this reason, the algorithm must operate effectively with as little processing power as possible, and be able to quickly produce a working solution, which may later be improved if the AI is allowed more time.

IV. Micromanagement Task Allocation System

The proposed MTAS that we present in this section translates the game world, and the objectives handed to each group, into a set of tasks. The most prominent feature in this system that sets it apart from other task assignment strategies is that each instance of a task manages the agents assigned to it. This means that, instead of each agent reasoning about how to accomplish the task in a decentralized manner, the task coordinates the actions of the agents assigned to it centrally. This facilitates cooperative behavior, and minimizes the impact of agents being destroyed, as the task persists even if there are no agents currently assigned to it. Using tasks as the basis of the AI also makes implementation more simple and flexible, and makes the AI more extensible [36].

A. The Task-Based Micromanagement System

All of the major components of the micromanagement AI system can be seen in Fig. 2. The event handler creates tasks based on events in the game world (e.g., sighting enemy units, units taking damage, unit death, construction completion, etc.), or updates existing tasks if the event is relevant to a previously created task. Alternatively, a group can request that the event handler create a goal-based task (such as "move to location X") when the group is given a goal by the strategy-level AI to take up position at location X.

The role of a group is to calculate a priority for each current task, calculate the proficiency of each unit for each task, and assign a weighting that represents how well this task aligns with the group's current goals (details are covered in Section IV-C).

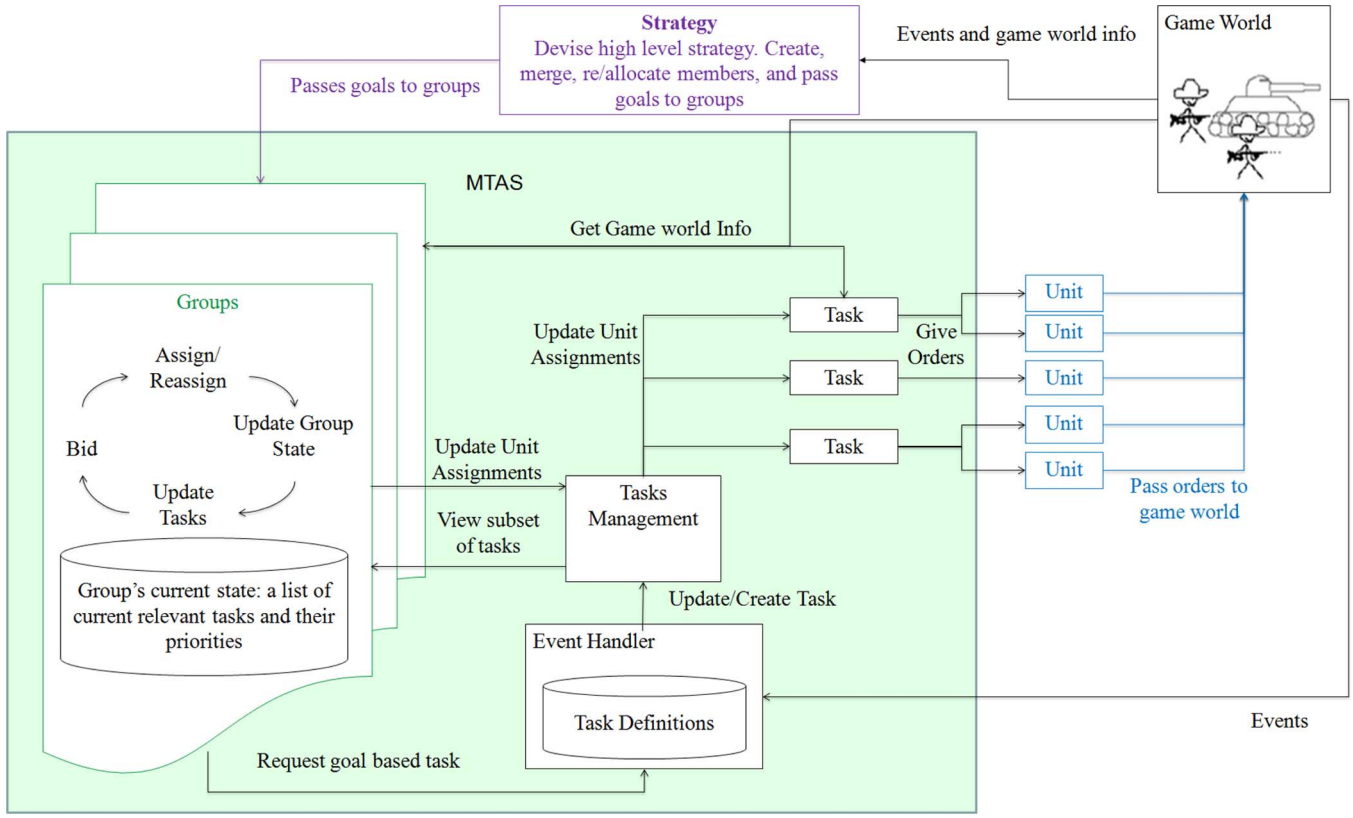

Fig. 2. MTAS architecture and its interface with existing strategy level and unit components within a full RTS AI.

Based on these values, the group assigns its members to the best task at the time. As a point of optimization, and/or to provide some additional control over which tasks members of the group are able to be assigned to, the group can filter tasks based on the abilities and goals of the group or the task's position (e.g., only allow members to bid on tasks within a given radius of the location of the group's objectives). This reduces the severity of the exponential growth in the number of bids calculated with respect to the number of units and tasks. Each task then takes the group members assigned to it (which may be from many different groups) and issues the relevant instructions to each of them. This cycle of update, bid, reassign, execute is processed as frequently as the AI budget allows.

This system is able to react quickly in the fast paced, highly dynamic, and unpredictable micromanagement environment by reevaluating each unit's bids on each of the current tasks, and reassigning units to the best possible task at that moment.

B. Tasks

In any RTS, there are a finite number of tasks that can be issued to units (e.g., attacking, moving, patrolling, defending, repairing, etc.). Ideally, the set of tasks should encompass all potential possibilities, but, in practice, this is infeasible. For example, tasks that target a location (such as move to location, patrol, or build structure) can potentially target thousands of different locations, most of which are nonsensical. Therefore, restrictions need to be placed on the creation of location based tasks by restricting the number of locations explored.

For each unit order type, a different task template needs to be defined. As each task is a self-contained object that describes how it is created, maintained, executed, and completed, the template for a task type must contain each of the following:

- a list of events that may trigger the creation of the task;
- a list of abilities a unit must have to contribute to the task;
- min/max/desired quota of units for the task;
- a bidding function that calculates the utility of any unit partaking in the task using a sum of weighted numerical values related to the current state of the task and unit in question;
- the orders that will be issued to units partaking in this task;
- an update procedure that maintains all the game information relevant to the task (information relevant to issuing orders to units and the bidding function);
- completion/invalidation conditions that define when the task should no longer be pursued.

As an example, consider the task "attack flying unit x," which represents the need to attack an enemy flying unit. The creation of the task might be triggered by the event of an enemy flying unit being sighted or attacking an ally. The critical requirements for a unit to be considered for this task would be the ability to attack flying units. The maximum unit quota could be the number of units required in order for the sum of their attack strengths to equal or exceed the strength of the target, as this should result in approximately one shot from each unit destroying the target. The minimum unit quota would likely be one, as there is no minimum number of units required to make this task practical. The bidding function would weigh up relevant factors such as the strengths of units in the group, their distance from the target, the importance of destroying the enemy unit, and the degree of threat the enemy unit poses to the group's current objectives.

Unit behavior under this task may involve some maneuvering to get the enemy in range, and then giving the unit an order to attack the target. The task would be deemed complete if the target is destroyed, and may be deemed invalid if the enemy unit moves out of sight for a period of time. When it comes to updating the task, details such as the enemy unit's position and health would change over time and, thus, need to be periodically updated. This, in turn, leads to the need to recalculate the bids from units in the group to take these changes into account. This may result in units being reassigned to different tasks or it may strengthen the bid for the task.

C. Bidding Functions

The system allocates tasks to units based on bidding functions that are similar to the multiagent system concept of utility functions [34]. These bidding functions calculate the utility of assigning any given unit to any task through a weighted sum of contributing factors such as distance, unit health, whether allies outnumber enemies, or *vice versa*, etc.

The bidding function associated with a task takes the form of a weighted sum of factors deemed relevant to the decision to allocate a unit to that task. Weighted sum systems are a comprehensible and expressive tool for decision making, allowing not just the AI to make intelligent decisions, but also allowing a human to gain insight into the decision-making process by inspecting the weights assigned to each factor [37]. The factors are grouped into two categories, referred to as proficiency and priority. The proficiency factors (such as the unit's distance to the target, its speed, and its range) gauge how useful the unit would be in completing this task, while priority factors (such as threat to the group, and distance from the group's current position) measure how important it is that this task be completed. Ideally, once a suitable set of data values are extracted from the game state and appropriate weights applied to each of them, the function should lead to the assigning of important actions (i.e., tasks) to the units best suited to completing them.

As an example, consider the bidding function for an "attack flying unit" task. Factors relevant to assessing a unit's proficiency to perform this task might include the time it would take to intercept the target (given the target's position and current velocity, and the bidding unit's position and top speed); the threat A poses to B (i.e., how much damage it could do); and whether the unit is intended for anti-air (AA) combat. Thus, unit A's bid for the task "attack flying unit B" would be calculated as follows:

$$\begin{aligned}\text{Proficiency} = {} & W1 \times \text{TimeToIntercept}(A, B) \\ & W_2 \times \text{Threat}(A \rightarrow B) + W_3 \times \text{IsAAUnit}(A). \end{aligned} \quad (1)$$

The priority is calculated by the group, based on its current state and general objectives, such as self-preservation, and may include criteria to favor certain unit types (such as workers) and buildings (such as resource buildings). For the example task, these could include the number of units currently attempting the task (to promote ganging up on a few units at a time, thus preventing the group from dispersing to attack many targets), the threat B poses to the group, and the trajectory of the unit relative to the group's location, to favor attacking units that are stationary or moving toward the group

$$\begin{aligned}\text{Priority} = {} & W_1 \times \text{NumberOfUnitsAssigned}() \\ & + W_2 \times \text{Threat}(B \rightarrow \text{Group}) \\ & + W_3 \times \text{Trajectory}(B \rightarrow \text{Group}). \end{aligned} \quad (2)$$

The final bidding function is defined as the sum of proficiency and priority, scaled by a weighting W_g that represents how well the task aligns with the group's current goal, and scaled by a commitment value W_c that increases the bid for the task this unit is currently assigned to in order to prevent switching between tasks too rapidly

$$\text{Bid} = W_g \times W_c \times (\text{Proficiency} + \text{Priority}). \quad (3)$$

If the group has no current goals assigned to it from the strategy-level AI, then $W_g = 1.0$. Otherwise, bids for the primary task (i.e., the task matching the current goal) receive a small boost ($W_g = 1.2$), while similar or complementary tasks receive a smaller boost ($W_g = 1.1$), and tasks that are contrary to the goal are penalized ($W_g = 0.8$).

D. Task Assigning/Reassigning Heuristic

The mapping of potential tasks to potential units is a many-to-many relationship: for each task, there may be many possible units that could attempt to complete it, and for each unit, there may be many tasks that it could attempt to complete.

As discussed earlier, the number of available task assignments will usually exceed the number of units. This means that once all units are assigned to a task, there will be many tasks with vacancies. So by exploring the solution space from the perspective of assigning each unit to a task, fewer options should need to be explored.

Due to processing constraints and the need to continuously reevaluate task assignments, the TA heuristic needs to be able to produce a workable solution near instantly. With this in mind, a two-stage operation is proposed. The first stage is to restrict the number of tasks each group presents to its members. This could be based on how long it would take to get to that task's location, and/or a rule base that allows different types of groups to pursue different types of tasks (i.e., bomber squadrons only accept "attack building" tasks). This provides more direct control over the group behaviors, and also reduces the exponential growth in the number of bid calculations that occurs if all units bid for all tasks.

The second stage involves running the following heuristic.

```
Order unassigned units by priority
FOR each unit
  Sort unit's task list by bid value greatest to smallest
  IF the unit is not assigned to the first task and the task is
  not full THEN
    Assign unit to task
  ELSE
    Add unit to list of units not in first preference
    FOR each other task
      IF task not full THEN
        Assign unit to task
```

```
            Break loop
          End IF
        End FOR
      End IF
    End For
    WHILE list of units not in first preference is not empty
      FOR each unit in list of units not in first preference
        FOR each task higher than the currently assigned task
          IF this unit's bid > a unit currently assigned to the task
          THEN
            Assign this unit to the task
            Unassign the other unit
            Add the other unit to the list of agents not in first
            preference
            FOR each of the unassigned unit's bids
              IF task not full THEN
                Assign agent to task
                Break loop
              End IF
            End FOR
            Break loop
          End IF
        End FOR
        Remove this agent from the list of agents not in first
        preference
      End FOR
    End WHILE
```

This heuristic was developed to satisfy a number of desirable features. The algorithm is simple, and can be broken out of with a solution that has every agent assigned to an action. The algorithm should not suffer from infinite loops as it is a hill climber, trading agents out of tasks only when an agent with a higher bid is found. Because the search space is limited to units that are not currently assigned to their best task, the algorithm only has to work hard when bid values or the task set has changed significantly since the last reassignment.

E. Integrating MTAS Into an RTS AI

In terms of the layered RTS AI approach, MTAS rests predominantly within groups, but effectively introduces a new layer: tasks. Groups are responsible for assigning their members to the best task based on the bidding functions, but once assigned, the task takes control of the unit and issues the appropriate orders. So far as the strategy-level AI is concerned, the group could be assigned one or many goals and would report back to the strategy level upon success or failure to complete the goal. Following this structure, the only thing required of the strategy-level AI is the ability to create and manage groups, and issue goals to those groups. Any AI that has been implemented using the layered approach should already satisfy this requirement, thus theoretically enabling MTAS to be integrated.

V. Implementation

To evaluate the potential of MTAS, it was integrated into an existing AI, E323AI, written for the Spring RTS engine. E323AI is capable of reliably defeating many other available AI's written for the Spring RTS engine in the *Balanced Annihilation* and XTA RTS games.

The Spring RTS engine was chosen for several reasons. First, Spring supports a number of RTS games, all of which share the same AI interface. This allows developers to create AI players in a range of languages (including Java, C++, Python, and Lua) and to evaluate these players on many different games. Many open source games and AIs are available from the Spring web site. Second, the Spring engine also allows for purely AI versus AI battles that can be watched by human observers, and also supports Lua widgets that can be used to collect data or provide additional visual feedback and user interface features. This allows for real-time visual feedback debugging and behavioral analysis; streaming of in-game data to log files for debugging, behavioral analysis, and statistical analysis; and use of interfaces that can display and edit any configurable settings in both the AI or widget.

The RTS game *Balanced Annihilation* was selected because it is a popular, stable, and long running game on the Spring engine, and has many compatible AIs.

Based on previous trials that we had conducted on *Balanced Annihilation*, E323AI clearly outperforms the other available AIs. This is probably due to its fine-tuned buildup sequence and relatively superior micromanagement, which enable it to cripple the opponent in very early attacks. Although comparatively superior to other existing micromanagement systems, the micromanagement in E323AI currently relies on a complex web of semihierarchical "if this do this" conditions that have been individually programmed by the AI designers. Since E323AI already boasts a tolerable level of micromanagement, and is clearly the strongest AI for the *Balanced Annihilation* game, it was deemed to be the most appropriate AI in which to implement MTAS.

MTAS was successfully integrated into E323AI's hierarchical AI structure. With one exception, this was achieved without having to alter the strategy level's interface with the groups. E323AI's strategy level passes instructions to groups that include: attack unit, move to location, construct building, guard object, and repair object. In the original E323AI, the unit group component would receive these instructions and simply broadcast them to all members of the group. With MTAS integrated, these instructions are intercepted, and the group component no longer gives orders directly to units. Instead, the details of the instructions are used to create an equivalent "task" which is set to being the primary task, and the current state of the group is set based on the type of instruction (i.e., if the group receives an instruction to attack unit x, a task for attacking unit x is created and becomes the primary task, with the group now in an attack state).

The one alteration to the interface between strategy and groups was that MTAS required the strategy level to make groups aware of their classification (economic, attack, antagonize, or scout). These classifications affect the group's selection of local tasks, and determine how aggressive the group is.

TABLE I
VICTORY RATE FOR E323AI/MTAS V E323AI ORIGINAL

Map Name	Games Played	Wins	Wins (%)	95% CI for wins (%) *	
Comet Catcher Redux	800	430	53.8	50.3	57.2
Altored_Divide	800	467	58.4	54.9	61.7
Titan-v2	800	468	58.5	55.1	61.9
DeltaSiegeDry	800	526	65.8	62.4	69.0
FolsomDamFinal	800	578	72.3	69.0	75.2
All Maps	4000	2469	61.7	60.2	63.2

* 95% confidence interval was calculated using a Z test

TABLE II
CLASSIFYING TYPE AND REASON FOR VICTORY

	E323AI/ MTAS	% of Victories	E323AI Original	% of Victories
Type of Victory				
Dominated	1388	56.2	830	54.2
Turn Around	1081	43.8	701	45.8
Reason for Victory				
Assassination	31	1.3	13	0.8
Production	756	30.6	607	39.6
Military	603	24.4	297	19.4
Productions & Military	1076	43.6	606	39.6
Base Destruction	30	0.1	8	0.5

Previously, this information was only available to economic and military components.

All the factors and weights included in the bidding functions used in this implementation were constructed using a combination of domain knowledge and trial and error. An initially large set of potential "contributing factors" were calculated from data extracted from the game. Weights were then adjusted in two stages. First, adjustments were made based on observed behavior over many RTS games until little or no observable ineffective behavior could be seen. Second, the weights were adjusted over successive rounds of trials until an acceptable win/loss ratio was achieved. Many of the initial set of contributing factors were actually assigned weights of 0 in the final round of testing, effectively excluding them from the final set of contributing factors.

VI. RESULTS AND DISCUSSION

To evaluate MTAS, a series of one versus one RTS matches were run, with one player controlled by the original E323AI and the other controlled by the E323AI edited to include MTAS, which we will refer to as E323AI/MTAS. These matches were run over five maps (Comet Catcher Redux, Altored Divide, Titanv2, Delta Siege Dry, and Folsom Dam Final). Four different scenarios were set up in each map such that E323AI/MTAS would start in one of the four corners of the map and E323AI Original would start in the opposite corner. The victory condition was set to "Kill Commander."

Table I shows the victory rate over a total of 4000 games (800 games from each of the five maps). Over the 4000 games, E323AI/MTAS was victorious 2469 times. The significance of this 61.7% victory rate was evaluated using a Z-test. For a H0 of $p = 0.602$, $Z = 1.96$, which translates to a confidence level of 0.025. According to this assessment, there is no doubt that E323AI/MTAS outperforms the original E323AI, and we can be more than 97.5% confident that it has a victory percentage greater than 60% (i.e., a win/loss ratio of at least 3:2).

It is interesting to note the significant difference in the victory rates between the first three maps and the last two. This is most likely due to the fact that the simple spatial reasoning used in the current implementation takes into account path distance, but not the steepness of the terrain. The first three maps contain many steep slopes which, while still traversable, slow units down dramatically. The last two maps contain much more defined cliffs and plains that either have no effect on speed, or are completely path blocking. Whatever the reason for this discrepancy, it would suggest that, while it is possible to find a set of weights that generally improve performance over all maps, what works well in one map may not work so well in another.

It is also useful to compare the differences between E323AI/MTAS and E323AI in terms of both the type of victory achieved, and the reason for the victory. Victory types were split into two categories: dominated (winning team dominated throughout game) and turnaround (winning team recovered from being at a disadvantage). Five possible reasons for victory were considered: assassination (winner killed opposing commander); production (winner produced more structures and units than opponent), military (winner utilized military more effectively), production and military (both were contributing factors), and base destruction (destruction of non-military infrastructure such as factories, workers, etc.). Criteria for classifying the games into the above categories are based on the concept of net worth. (See the Appendix for definitions and classification criteria.)

Table II provides the victory type and victory reason data for the 4000 games.

Looking at the reasons for victory, E323AI/MTAS appears to have more purely military-based victories (24.4% compared to 19.4%) while E323AI Original has more purely production-based victories (39.6% compared to 30.6%). The important note here is that, because the reason for victory can be both, getting a military victory means the loser actually had superior production and *vice versa*. So this would suggest that the current set of tasks and weights is showing its greatest effect in improving combat micromanagement. This is not surprising as base building and resource gathering is predominantly strategic in this RTS.

Irrespective of the type or reason for a victory, the superior performance of MTAS is due to the ability it provides for groups to multitask. We now provide some observable examples of how group multitasking appears to contribute to this superior performance.

The first example pertains to group formation. E323AI creates groups by initializing an empty group, and then adding members one by one as they are produced from a factory. In the original implementation of E323AI, these groups and their members are left without instruction until the group is deemed

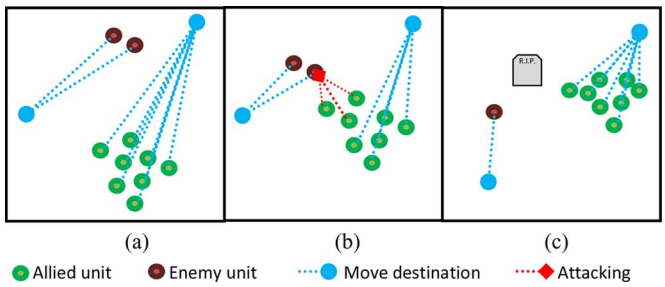

Fig. 3. Example of MTAS group multitasking: (a) allied units moving to new location; (b) three allied units divert to attack enemy scout group; and (c) allied units resume to moving to new location.

fully formed. With MTAS, implemented members of these (forming) groups are able to defend themselves and the local region by attacking enemy units that enter the group's region of interest and returning to the location near the factory if the enemy is destroyed or flees.

A more interesting example concerns tasks such as that of destroying enemy scout groups, which pose a significant threat to resource gathering efforts, and is depicted in Fig. 3. The new behavior that is promoted by weights that increase bids for attack tasks if allied units outnumber enemy units in the region is most obviously observed when the group is moving to a new location. In the original E323AI, units ordered to move to a new location will only attack an enemy if they can attack and move all at the same time and the enemy unit strays within range. With MTAS in place, some or all of the group members can temporarily switch to an attack task and be given orders to attack units that are beyond their range. Only the attack tasks will be pursued, while it does not require much of a diversion from their path. Once the scouts are destroyed or stray too far from the group, these units return to moving to the group's new location.

While experimenting with weights in the bidding function, the most significant challenge was preventing units from oscillating between tasks. This occurs for two reasons: 1) the bidding function for one or more tasks increases and decreases too rapidly in response to variables in the world that fluctuate; or 2) the act of pursuing a task decreases the bid for that task, or increases the bid for a task that would result in an opposing action. This fact came to light in the distance factors for the "move to group's location" tasks.

Initially, the "move to group's location" task contained a weight that increased the unit's bid for the task as it got further away from the group. This resulted in units assigned to tasks that required them to travel away from the group's location turning back halfway when they switched to the move tasks because the bid overtook that of the original task. Then, once close to the group's location, they would be reassigned to the original task once the bid for moving to the group's location fell below the bid for the original task. This disastrously inefficient behavior was rectified by setting the "move to group's location" bid to be a constant value while all other tasks receive a negative weighting for their distance from the group's location and from the unit in question. So, with these new weights, the act of pursuing a task increases its bid and has no effect on the move to location bid.

Unfortunately, in the current implementation, there exist complex or extreme situations where MTAS does not always follow orders. This is because the bidding functions suggest that current conditions are particularly unfavorable for pursuing the goal task, and E323AI does not distinguish between trivial and important goals. If the strategy component were equipped to transfer importance and/or urgency information, then MTAS would be more than capable of pursuing tasks that are unfavorable from the group's local perspective.

Adjusting the bidding function weights manually is both time consuming, and less likely to find interesting or unexpected positive behaviors. The obvious solution would be to incorporate automated learning, capable of finding strong sets of weights. As MTAS uses sets of weights that could easily be interpreted as a genetic code, genetic algorithms could be employed to evolve a population of these sets. The initial population could be generated from a combination of manually and randomly assigned weights. The fitness function could incorporate any of a number of performance metrics, such as victory ratio, goals completed per group, and/or timely completion of goals. The population could then be evolved over successive generations of round-robin style games. In addition to just finding one strong set of weights, speciation algorithms could potentially uncover a number of different but equally successful sets of weights.

VII. Conclusion

This paper provides a definition of the group level micromanagement problem in RTS from the perspective of a TA system, establishing our understanding of the environment's characteristics and desired system qualities in relation to existing TA problems. MTAS is presented as a flexible, understandable, and generally applicable model for providing an intelligent allocation of tasks within a diverse group of units in pursuit of goals passed to the group from the strategy-level component.

The main advantage that MTAS brings to RTS AI over current techniques is a system that can micromanage groups of units that is flexible enough to be applicable to a wide variety of unit types, can pursue goals set by the strategic component of the RTS AI, provide meaningful feedback in regards to goal completion/failure/invalidation, and behave tolerably in the absence of explicit goals. Groups that can multitask in an intelligent, nonpredefined manner opens up the possibility of unit groups containing a wider variety of unit types, as members of the group can perform different actions that play to their individual strengths and abilities, and unit type assemblages are not restricted to those that the AI programmer has accounted for in advance (as is often the case when using scripted group behavior or other tailored micromanagement solutions).

The fact that MTAS was successfully retrofitted into E323AI without altering the interface between strategy and groups shows that it is possible to integrate it to an AI that follows the hierarchical approach. Thus, MTAS should be able to be used in conjunction with any strategy-level reasoning component and improve the overall proficiency of RTS AI, irrespective of the strategy level's internal implementation. These results demonstrate that there are definitely gains to be had by improving micromanagement in RTS AIs, and that MTAS is a viable model.

The structure of MTAS is such that many of the key components (such as task creation/filtering, bid function mechanics, assignment heuristics, and establishing the bid function values and weights) could be improved by incorporating more elaborate techniques. The techniques used in the current implementation proved to be effective, and have the advantage of being easily understandable, but are relatively simplistic. As such, there are many possible future developments that could enhance the effectiveness of MTAS: as mentioned earlier, learning algorithms could be used to find effective weights for the bid functions by evolving a population of competing sets of weights; the weighted sum bidding functions could be replaced with multiple-input–single-output feedforward neural networks that could potentially be evolved to capture more complex relationships between inputs; influence maps representing enemy strength, allied strength, cover, etc., could be used to extract values for the bidding functions; and more elaborate special reasoning taking into account unit ranges, movement, and terrain could facilitate more specific task definitions, such as attack unit from specific location or zone. Alternatively, the implementation of the tasks themselves could be enhanced using the unit level techniques that have been shown to provide more effective unit behaviors. For example, Synnaeve and Bessiere's [28] Bayesian reasoning system could be used to coordinate units attacking and moving while completing the task.

Appendix

Calculation of net worth is based on the "metal" cost of each unit and structure that a player gained or lost. These data were collected during each match and recorded at intervals of approximately one in-game minute. These data included, from both the winner (W) and the loser (L), the following:

- the comparative net worth (CNW) of each player (the player's net worth divided by the combined net worth of both players);
- net worth military constructed (NWMC);
- net worth military lost (NWML);
- net worth other constructed (NWOC);
- net worth other lost (NWOL).

These data were used to classify the type and reason for the victory in accordance with the following.

A. Victory Types

1) Dominated: the winning player's CNW never fell below 0.48.
2) Turnaround: the winning player's CNW did fall below 0.48, but they managed to turn the game around.

B. Reason for Victory

The data used for discerning the reason for victory were recorded at the last point in the game where net worth of both players still lay between 0.45 and 0.55. The rationale for this is that, up until this point, both players have had approximately the same amount of resources to work with at any one time, and at the 9:11 net worth ratio, the player with a net worth of 0.55 nearly always wins. Reasons for victory were explored in the following order: assassination, military and/or production, and, finally, base destruction.

1) Assassination: The winning player managed to kill the opposing commander even though their net worth was less than that of the losing player.
2) Military: The victory was put down to superior use of military forces if the ratio of the damage inflicted by the winner to damage received was greater than the ratio of the winner's army to the loser's army

$$\frac{\mathrm{NWMC}_W}{\mathrm{NWMC}_L} < \frac{\mathrm{NWML}_L + \mathrm{NWOL}_L}{\mathrm{NWML}_W + \mathrm{NWOL}_W}. \tag{4}$$

3) Production: The victory was put down to superior production if the winner produced more military and nonmilitary units and structures than the loser

$$\mathrm{NWMC}_L + \mathrm{NWOC}_L < \mathrm{NWMC}_W + \mathrm{NWOC}_W. \tag{5}$$

4) Base destruction: The destruction of nonmilitary infrastructure such as factories, resource gathering structures, and workers

$$\mathrm{NWOL}_W < \mathrm{NWOL}_L. \tag{6}$$

References

[1] D. Thomas, "New paradigms in artificial intelligence," in *AI Game Programming Wisdom*, S. Rabin, Ed. Newton Center, MA, USA: Charles River Media, 2004, vol. 2, pp. 29–39.

[2] M. Ponsen, P. Spronck, H. Muñoz-Avila, and D. W. Aha, "Knowledge acquisition for adaptive game AI," *Sci. Comput. Programming*, vol. 67, pp. 59–75, 2007.

[3] D. H. Cerpa and J. Obelleiro, "An advanced motivation-driven planning architecture," in *AI Games Programming Wisdom.*, S. Rabin, Ed. Newton Center, MA, USA: Charles River Media, 2008, vol. 4, pp. 373–382.

[4] K. Mishra, S. Ontañón, and A. Ram, "Situation assessment for plan retrieval in real-time strategy games," in *Advances in Case-Based Reasoning*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2008, vol. 5239, pp. 355–369.

[5] K. T. Andersen, Z. Yifeng, D. D. Christensen, and D. Tran, "Experiments with online reinforcement learning in real-time strategy games," *Appl. Artif. Intell.*, vol. 23, pp. 855–871, 2009.

[6] S. Bakkes, P. Spronck, and J. van den Herik, "Rapid and reliable adaptation of video game AI," *IEEE Trans. Comput. Intell. AI Games*, vol. 1, no. 2, pp. 93–104, Jun. 2009.

[7] S. C. J. Bakkes, P. H. M. Spronck, and H. Jaap van den Herik, "Opponent modelling for case-based adaptive game AI," *Entertain. Comput.*, vol. 1, pp. 27–37, 2009.

[8] B. Weber and M. Mateas, "Conceptual neighbourhoods for retrieval in case-based reasoning," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science, L. McGinty and D. Wilson, Eds. Berlin, Germany: Springer-Verlag, 2009, vol. 5650, pp. 343–357.

[9] B. G. Weber, M. Mateas, and A. Jhala, "Learning from demonstration for goal-driven autonomy," in *Proc. 26th AAAI Conf. Artif. Intell.*, 2012, pp. 1176–1182.

[10] P. Ng, S. Shiu, and H. Wang, "Learning player behaviours in real time strategy games from real data," in *Rough Sets, Fuzzy Sets, Data Mining and Granular Computing*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2009, vol. 5908, pp. 321–327.

[11] G. Synnaeve and P. Bessiere, "A Bayesian model for opening prediction in RTS games with application to Starcraft," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011, pp. 281–288.

[12] M. Z. Rashad, "A rough—Neuro model for classifying opponent behaviour in real time strategy games," *Int. J. Comput. Sci. Inf. Technol.*, vol. 4, p. 11, Oct. 2012, 2012.

[13] D. Churchill and M. Buro, "Incorporating search algorithms into RTS game agents," presented at the Artif. Intell. Adversarial Real-Time Games Workshop, Stanford, CA, USA, 2012.
[14] A. Dahlbom and L. Niklasson, "Goal-directed hierarchical dynamic scripting for RTS games," in *Proc. 2nd Artif. Intell. Interactive Digit. Entertain. Conf.*, Marina Del Rey, CA, USA, 2006, pp. 21–28.
[15] M. Freed, T. Bear, H. Goldman, G. Haytt, P. Reber, and J. Tauber, "Towards more human-like computer opponents," in *Proc. AAAI Spring Symp.*, 2000, pp. 22–26.
[16] P. Avery and S. Louis, "Coevolving influence maps for spatial team tactics in a RTS game," presented at the 12th Annu. Conf. Genetic Evol. Comput., Portland, OR, USA, 2010.
[17] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Case-based planning and execution for real-time strategy games," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science, R. Weber and M. Richter, Eds. Berlin, Germany: Springer-Verlag, 2007, vol. 4626, pp. 164–178.
[18] U. Jaidee, H. Muñoz-Avila, and D. Aha, "Learning and reusing goal-specific policies for goal-driven autonomy," in *Case-Based Reasoning Research and Development*, ser. Lecture Notes in Computer Science, B. Agudo and I. Watson, Eds. Berlin, Germany: Springer-Verlag, 2012, vol. 7466, ch. 15, pp. 182–195.
[19] C. Hermann, H. Melcher, S. Rank, and R. Trappl, "Neuroticism—A competitive advantage (also) for IVAs?," in *Intelligent Virtual Agents*, ser. Lecture Notes in Computer Science, C. Pelachaud, Ed. Berlin, Germany: Springer-Verlag, 2007, vol. 4722, pp. 64–71.
[20] P. Huo, S. Shiu, H. Wang, and B. Niu, "Case indexing using PSO and ANN in real time strategy games," in *Pattern Recognition and Machine Intelligence*, ser. Lecture Notes in Computer Science. Berlin, Germany: Springer-Verlag, 2009, vol. 5909, pp. 106–115.
[21] P. H. F. Ng, Y. J. Li, and S. C. K. Shiu, "Unit formation planning in RTS game by using potential field and fuzzy integral," in *Proc. IEEE Int. Conf. Fuzzy Syst.*, 2011, pp. 178–184.
[22] F. Paanakker, "Risk-adverse path finding using influence maps," in *AI Games Programming Wisdom*, S. Rabin, Ed. Newton Center, MA, USA: Charles River Media, 2008, vol. 4, pp. 173–178.
[23] J. Hagelback and S. J. Johansson, "A multi-agent potential field-based bot for real-time strategy games," *Int. J. Comput. Games Technol.*, 2009, Article ID 910819.
[24] C. Jurney, "Company of heroes squad formations explained," in *AI Games Programming Wisdom*, S. Rabin, Ed. Newton Center, MA, USA: Charles River Media, 2008, vol. 4, pp. 61–70.
[25] I. Gabriel, V. Negru, and D. Zaharie, "Neuroevolution based multi-agent system for micromanagement in real-time strategy games," presented at the 5th Balkan Conf. Inf., Novi Sad, Serbia, 2012.
[26] S. Wender and I. Watson, "Applying reinforcement learning to small scale combat in the real-time strategy game StarCraft:Broodwar," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 402–408.
[27] N. Othman, J. Decraene, C. Wentong, H. Nan, M. Y. H. Low, and A. Gouaillard, "Simulation-based optimization of Starcraft tactical AI through evolutionary computation," in *Proc. IEEE Conf. Comput. Intell. Games*, 2012, pp. 394–401.
[28] G. Synnaeve and P. Bessiere, "A Bayesian model for RTS units control applied to Starcraft," in *Proc. IEEE Conf. Comput. Intell. Games*, 2011, pp. 190–196.
[29] D. Churchill, A. Saffidine, and M. Buro, "Fast heuristic search for RTS game combat scenarios," in *Proc. 8th AAAI Conf. Artif. Intell. Interactive Digit. Entertain.*, Palo Alto, CA, USA, 2012, pp. 112–117.
[30] M. Buro and D. Churchill, "Real-time strategy game competitions," *AI Mag.*, vol. 3, pp. 106–108, 2013.
[31] T. C. Lueth and T. Laengle, "Task description, decomposition, and allocation in a distributed autonomous multi-agent robot system," in *Proc. IEEE/RSJ/GI Int. Conf. Intell. Robots Syst./Adv. Robot. Syst. Real World*, 1994, vol. 3, pp. 1516–1523.
[32] P. Dasgupta, "Multi-robot task allocation for performing cooperative foraging tasks in an initially unknown environment," in *Innovations in Defence Support Systems*, ser. Studies in Computational Intelligence. Berlin, Germany: Springer-Verlag, 2011, vol. 338, pp. 5–20.
[33] K. Macarthur, R. Stranders, S. Ramchurn, and N. Jennings, "A distributed anytime algorithm for dynamic task allocation in multi-agent systems," in *Proc. 25th Conf. Artif. Intell.*, San Francisco, CA, USA, 2011, pp. 701–706.
[34] R. G. Smith, "The contract net protocol: High-level communication and control in a distributed problem solver," *IEEE Trans. Comput.*, vol. C-29, no. 12, pp. 1104–1113, Dec. 1980.
[35] D. Trentesaux, R. Dindeleux, and C. Tahon, "A multicriteria decision support system for dynamic task allocation in a distributed production activity control structure," *Int. J. Comput. Integr. Manuf.*, vol. 11, pp. 3–17, 1998.
[36] A. J. Champandard, "Getting started with decision making and control systems," in *AI Games Programming Wisdom*, S. Rabin, Ed. Newton Center, MA, USA: Charles River Media, 2008, vol. 4, pp. 257–264.
[37] W. Falke and P. Ross, "Dynamic strategies in a real-time strategy game," in *Genetic and Evolutionary Computation—GECCO 2003*, ser. Lecture Notes in Computer Science, E. Cantú-Paz, Ed. Berlin, Germany: Springer-Verlag, 2003, vol. 2724, pp. 211–211.

Keith D. Rogers received the B.S. degree in computer science in games technology (with first class honors) from the School of Engineering and Mathematical Sciences, Latrobe University, Melbourne, Vic., Australia, in 2009, where he is currently working toward the Ph.D. degree in artificial intelligence.

Andrew A. Skabar received the Ph.D. degree in electrical and electronic systems engineering from Queensland University of Technology, Brisbane, Qld., Australia, in 2001.

He worked at the International University, Bruchsal, Germany, from 2001 to 2002, and then at Deakin University, Waurn Ponds, Vic., Australia, from 2003 to 2004. Since 2005, he has been with the Department of Computer Science and Computer Engineering, La Trobe University, Melbourne, Vic., Australia. His general research interests lie in the area of pattern recognition and machine learning.