An Integrated Agent for Playing Real-Time Strategy Games

Josh McCoy and Michael Mateas

Expressive Intelligence Studio, University of California Santa Cruz
1156 High Street
Santa Cruz, CA 95064
mccoyjo@soe.ucsc.edu, michaelm@soe.ucsc.edu

Abstract

We present a real-time strategy (RTS) game AI agent that integrates multiple specialist components to play a complete game. Based on an analysis of how skilled human players conceptualize RTS gameplay, we partition the problem space into domains of competence seen in expert human play. This partitioning helps us to manage and take advantage of the large amount of sophisticated domain knowledge developed by human players. We present results showing that incorporating expert high-level strategic knowledge allows our agent to consistently defeat established scripted AI players. In addition, this work lays the foundation to incorporate tactics and unit micro-management techniques developed by both man and machine.

Introduction

Real-time strategy (RTS) games provide a rich and challenging domain for autonomous agent research (Buro 2003). The goal for players of RTS games such as the well-known Warcraft and Starcraft series is to build up armies capable of defeating enemy bases, while simultaneously defending one's base against enemy attacks. The object and action complexity, combined with real-time multi-scale play, provide a uniquely challenging domain for game playing agents.

RTS games contain a large number of unique domain objects and unique actions. Domain objects include different types of mobile units, buildings with varying defensive and unit production capabilities, building modifications, and resources that must be gathered, managed and spent in order to construct buildings and units. Actions include building and unit construction, choosing upgrades for buildings and units, resource management, and employing unit capabilities during battle. Action in an RTS occurs at multiple scale levels, such as high-level strategic decisions about which types of buildings and units to produce, intermediate tactical decisions about how to deploy groups of units across the map, and low-level micro-management decisions about individual unit actions. The combinatorics of this space of objects and actions precludes the use of game tree search-based techniques that have proven useful in board games such as chess. To make matters more complex, a successful RTS player must engage in multiple, simultaneous, real-time tasks. In the middle of a game, a player may typically be managing the defense and production capacities of one or more bases while being simultaneously engaged in several battles. Finally, RTS games often enforce incomplete information in the form of the "fog of war" that hides most of the map. The player can only see areas of the map where she has units, requiring the deployment of scout units to actively gather information about enemy activities.

These attributes of the domain argue for a game playing agent architecture that can incorporate human-level decision making about multiple simultaneous tasks across multiple levels of abstraction, and combine strategic reasoning with real-time reactivity. In this paper we present a novel agent architecture for playing RTS games. Our agent is decomposed into distinct competencies that mirror the competency distinctions made by expert human players, thus providing a framework for capturing and expressing human-level strategic, tactical and micro-management knowledge. Our results show that this approach can provide a level of play able to defeat two static strategies that have been used as benchmarks in the RTS research literature.

Related Work

Current research in game playing agents for RTS games has tended to focus on either the details of unit micro-management, or on high level strategy decisions that leave tactics and micro-management to the built-in unit AI. Although micro-management and strategy are certainly two of the competencies required for RTS play, the failure to build integrated agents has resulted in agents able to play only one small facet of the game or to not be able to play the game at competitive levels.

A number of researchers have focused on applying a single algorithm to a single facet of RTS game play: Monte Carlo planning over unit micro-management scenarios (Chung, Buro and Schaeffer 2005); PDDL used to explore the tactical decisions involved in building orders (Kovarsky and Buro 2006); and RMDPs used to generalize strategic plans (Guestrin et al. 2003). While each of these

systems provides local improvements, none are integrated, comprehensive agents capable of competently playing a full game.

Evolutionary learning on tactical decisions using dynamic scripting (Ponsen et al. 2006), and case-based reasoning over human player traces (Ontañón et al. 2007) b both take the next step by being capable of playing entire games. However, they each use a single component to do strategic, and limited tactical, reasoning. Unit micro-management is relegated to the simple-minded unit behaviors in the game engine. Additionally, while the case-based agent uses traces of human play to build a case library, the human players employed are weak compared to professional RTS players. Our agent uses expert knowledge gathered from professional players.

The SORTS agent is capable of playing an entire standard RTS game, including the use of high level strategy (Wintermute, Xu and Laird 2007). SORTS includes algorithms based on human perception to form unit groups and to focus attention. Unit micro-management is handled in the middleware with the use of finite state machines (FSMs). To enable a larger degree of tactical coordination between units, the FSMs that handles military and resource gathering units are managed by global coordinators. These coordinators employ simple learning to enhance the efficiency of the agent. High level strategic reasoning takes place in the Soar portion of the SORTS agent. While SORTS is an impressive system capable of playing complete games, and integrating multiple modules, improvements can still be made. Our agent adds the use of a reactive planning language capable of more tightly coordinating asynchronous unit actions in unit micro-management tasks, decomposes the agent into more distinct modules (domains of competence) and includes more expert human knowledge.

As we describe in this paper, strategic reasoning is fundamental to any agent whose goal is to competitively play RTS games. Furthermore, we show that there is a wealth of expert RTS knowledge that has yet to be leveraged in RTS AI research.

Expert RTS Play

Expert RTS play is as deeply skillful as expert chess play. Expert players and RTS communities have developed standard strategies, micro-management and tactical techniques. As in chess, part of expert play involves selecting techniques at multiple levels of abstraction in response to recognized opponent strategies, tactics and micro-management, and improvising within these techniques. But, as described in the introduction, the complexity of an RTS game (number of domain objects, parallel, asynchronous action, etc.) far exceeds chess. Even the physical skill involved in RTS play is daunting; expert human players routinely exceed 300 actions (distinct interface manipulations) per minute. The best players make handsome livings as full-time professionals in RTS leagues. In developing our integrated RTS player, we mined information available on expert techniques and rules of thumb (e.g. http://www.battle.net/war3/), as well as expert commentary on championship games (e.g. http://www.youtube.com/watch?v=IcPxu1RkeRU), and used this to determine the required competencies of our agent.

General rules of thumb have been distilled by the expert player community. Getting “behind on economy” almost guarantees a loss at expert levels. Like any rule of thumb, there are specific situations and strategies in which the rule is violated. For example, the “Probe Stop” involves halting economic expansion in favor of putting all available income into military production, causing a temporary spike in military strength. Managing the economy (collection and consumption of resources) in order to maximize mass-production of military and production units is a major subtask of expert play.

Human-developed unit micro-management techniques contain a subgroup that is applicable to nearly all RTS games. One of the basic micro-management techniques is dancing. Dancing is a specific use of ranged units in which a group of units attacks simultaneously, then “dances” back during their “cooldown”, the unit’s recovery period between attacks. This allows the ranged units to be as far away from the opposing units as possible during their cooldown period, without sacrificing offensive position.

Techniques such as dancing are called micro-management because they involve controlling the detailed movements of individual units. In the absence of micro-management, units respond to high-level directives (such as attack) using very simple built-in behaviors. Micro-management involves real-time decision-making on the part of the player and is another major subtask of the RTS player.

Tactics entails unit deployment and grouping decisions. Unlike micro-management, tactics involves coordinating groups of units to perform specific tasks. One tactic, commonly seen in the early game of Warcraft III (http://www.battle.net/war3/), involves coordinating units to block an enemy retreat, often using area effect attacks or blocking terrain bottlenecks with units. Tactical decision making, including a knowledge of common tactics and counter-tactics, is a significant subtask for the expert RTS player.

In RTS games, the map is only visible in a limited radius around friendly units. This requires active reconnaissance to gather information about the location and activities of enemy units. A common reconnaissance method early in the game is to send a worker around the map to find the enemy base. Base reconnaissance reveals the progress and nature of the economic build-up (revealing information about which strategies the enemy is likely following) and the physical layout of the base (e.g. is the base highly fortified against ground attacks). Reconnaissance is thus a significant subtask for the expert RTS player.

Finally, expert players develop and deploy high-level strategies. Strategies coordinate the style of economic buildup, the base layout, and the offensive and defensive

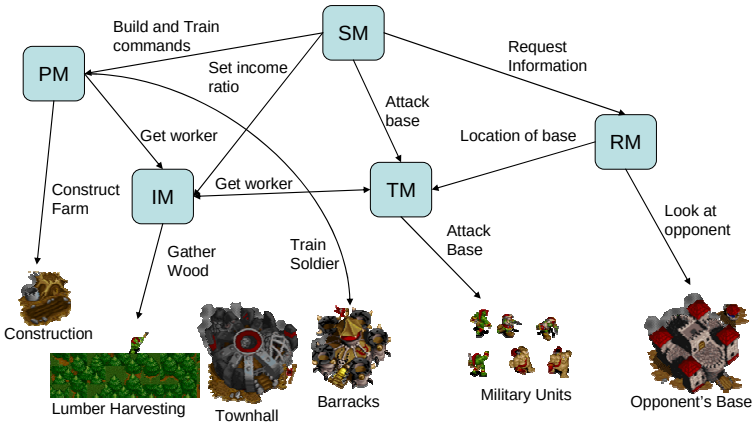

Figure 1 - This graph shows a small sample of the dependencies managers have on one another. The blue boxes are managers with abbreviated names (TM is tactics manager, RM is recon manager, etc). At the bottom of the graph are game entities found in Wargus.

style. The knight's rush is a strategy found in Warcraft II. The knight is a heavy melee unit available in the middle of the technology tree (the tree of dependencies between different unit and building types). In general RTS terms, a knight's rush is a heavy attack with units in the middle of the tech tree as soon as they are available. With this strategy, the economy is expanded as much as possible while researching the technology and constructing the buildings necessary for knights, with little to no resource spent on defensive units. This strategy leaves a player vulnerable until the ability to create knights is obtained, as early defense is traded for later offensive power.

Human experts decide on a high-level strategy very quickly at the beginning of the game, based on information such as the map size and number of opponents. This initial strategy selection determines the initial production order of buildings and units. For example, a small map favors a push for early military units, as attacks come earlier in the game. However, players must often switch strategy based on information gained through reconnaissance. Strategy determination, execution and switching is the last major subtask of the expert RTS player. The subtasks identified in our analysis of human expert play inform the managers of our integrated RTS agent.

Framework

The software framework of our agent consists of the ABL reactive planning language connected to the Wargus RTS engine.

ABL

ABL (A Behavior Language) is a reactive planning language similar in flavor to belief-desire-intention architectures such as PRS (Georgeff and Lansky 1987), though an ABL program does not require a commitment to a formal domain model. ABL serves as the glue for our integrated agent. Different distinct competencies in our agent, which can make use of distinct problem solving techniques, communicate with each other through ABL's working memory, and through ABL's dynamic subgoaling mechanism.

ABL, based on the Oz project believable agent language Hap (Bates, Loyall and Reilly 1992), adds significant features to the original Hap semantics, including first-class support for meta-behaviors (behaviors that manipulate the runtime state of other behaviors) and for joint intentions across teams of multiple agents (Mateas and Stern 2002).

Although ABL was originally designed to support the creation of autonomous believable characters, it has many features that work well in the space of RTS AI. Playing an RTS game requires the player to pay attention to many facets of the game simultaneously, reacting quickly and appropriately at multiple levels of abstraction including strategy, tactics and unit micro-management. Good RTS play requires a combination of deliberative, planful activity and real-time responsiveness to changing game conditions. ABL was precisely designed to combine reactive, parallel goal pursuit with long-term planfulness.

Wargus

Wargus is a clone of the game Warcraft II and its expansion, Tides of Darkness, both of which were developed by Blizzard Entertainment™. This clone is rendered in the open source RTS game engine Stratagus (Ponsen et al. 2005). The open source nature of Wargus allows integration with ABL and access to any part of the game state that Wargus itself has access to.

Another beneficial quality of Wargus is that it has an external AI scripting language, based on Lua, through which new agents can be added and existing agents can be modified. This scripting capability affords the creation and importing of a range of AI scripts with which to test our agent.

Agent Architecture

Our agent is composed of distinct managers, each of which is responsible for performing one or more of the major subtasks identified in our analysis of human expert play. As seen in Figure 1, the agent consists of income, production, tactics, recon, and strategy managers.

The first competency we have focused on is strategy, as strategic competence is required to play a complete game. Several of the other managers have been implemented with just enough functionality to provide an interface to the rest of the managers comprising the agent. By factoring our agent based on the study of expert play, we are able to easily modify individual managers and study the effect of increased or decreased competence of a specific manager on the overall strength of the agent.

Strategy Manager. The strategy manager is responsible for high level strategic decisions. The manager is composed of modules, which consist of behaviors implementing rules and associated ABL working memory elements.

The first task the strategy manager performs is to determine the proper initial order in which to construct buildings and units (the first 30 to 90 seconds of game play). The InitialStrategy module utilizes the recon manager to determine the distance between the agent's and opponent's starting locations on the game map, using thresholds to determine if the distance is small, medium, or large. Currently we use manually determined thresholds to make this determination, though there is certainly an opportunity for adaptation here. For small distances, the initial strategy involves constructing barracks and military units before the construction of a second farm (to support more workers); this allows the agent to defend against early attacks, as well as potentially perform an early attack, at the cost of an economy that grows slightly more slowly at the beginning. For medium distances, the initial strategy maximizes economic production; for separated enemies, there is time to build a robust economy and a significant military force before engaging in battles. This involves creating new workers without pause during the early game, and setting the default unit creation mix in the production manager to one worker to one soldier (the production manager will start creating soldiers when the preconditions for soldier creation, namely the construction of a barracks, have been fulfilled). Currently, for large distances, InitialStrategy performs the same behavior as for medium distances. This distinction between large and medium distances will be useful when the recon manager is sophisticated enough to determine suitable locations for the construction of additional bases; for small and medium distances, base attacks generally occur before there is time to build additional bases.

The next module, TierStrategy, is the highest priority recurring task in the strategy manager. "Tier" refers to the technology level achieved by the agent during economic production; different types of units become available as new building types are constructed (there are precondition relationships between building types that force a player to build up through the tiers). At each of the three tiers in Wargus, TierStrategy responsibilities include maintaining a unit control cap with regards to production capacity (i.e. construct the correct amount of farms), building a military/economic force superior to that of the opponent, and attacking when the agent has a military unit advantage. Our agent currently has a large amount of knowledge for tier one strategy, a small amount for tier two, and no knowledge for tier three. In future work, we will be expanding strategic competence for tier three, though there are no existing strong AI players we can test against for tier three.

TierStrategy starts making decisions after the initial building order controlled by InitialStrategy is complete. A primary responsibility for TierStrategy is determining which buildings and units are to be produced at any point in the game past the initial build order. For tier one, the agent grows economic and military unit production capacity as quickly as possible. To this end, the module continuously trains workers until the agent controls at least ten (unless InitialStrategy has primed the production manager to produce soldiers early, in which case soldier production is mixed in with worker production).

After the initial economic buildup, TierStrategy begins production of military capacity. Two barracks and a blacksmith are built sequentially after eight workers are available. After the first barracks has finished, the agent trains one soldier at the barracks for every new worker created. After all military production buildings are finished, the agent builds two soldiers for every worker. During all of this, TierStrategy monitors how many units can be supported given the current number of farms, ordering the construction of new farms to support increasing unit numbers.

At any time during tier one, three variations of the human developed "probe stop" strategy (in which all production shifts to military units) can occur. After twenty workers are created, the agent has enough income to continuously build grunts from both barracks while upgrading their offensive and defensive capacity via the blacksmith. At this point, only upgrades, soldiers, and farms to support new soldiers are created. Additionally, if the opponent has a lead of more than three military units, a probe stop is performed. When the agent has a deficit of more than two workers, the agent performs a reverse probe stop; all military production is halted in favor of ramping up resource gathering and production capacity.

As the game balance of Wargus is skewed toward heavy melee units, the tier two production strategy is to simply build a stables (the building which enables the construction of knights) and produce as many knights as possible.

TierStrategy is also responsible for determining when to attack, given the number of military units controlled by the agent vs. the opponent. This decision is currently made via empirically determined static thresholds: a 3 unit advantage in favor of the agent for tier one, 5 for tier two, and 8 for tier three.

Income Manager. The income manager is responsible for the details of controlling workers who gather resources, releasing workers for construction and repair tasks, and maintaining a gold to wood income ratio set by the strategy manager.

The first action performed by the income manager is to set the gold to wood income ratio given by the strategy manager. After the ratio is set, the manager is given workers, who have no assigned task, from the production manager. The newly assigned workers are placed in a resource list with regards to keeping the actual gold to wood income ratio closest to that set by the strategy manager.

The final responsibilities are to release workers for construction tasks (requested by the production manager) and repair tasks (by either the production or tactics

manager) and to put workers on the appropriate resource task if the strategy manager changes the income ratio.

Production Manager. The production manager is responsible for constructing units and buildings. At the heart of the production manager are modules that service three priority queues: a unit production queue, a building production queue, and a queue for repeated cycles of unit training and building construction.

Unit production pursues the training of the highest priority unit in the unit queue, ensuring that there are sufficient resources, and that the appropriate production building is free (e.g. barracks for training soldiers). Building production pursues the construction of the highest priority building, appropriately locking construction resources. This is necessary because simulation time passes between when the decision is made to build and a worker reaches the building location; without locking, resources would be spent elsewhere during this delay, causing the construction to fail.

Tactics Manager. The tactics manager takes care of unit tasks pertaining to multi-unit military conflicts. The tactics manager has three modules. The first assigns all military units to a unit group. The tactics manager provides an interface for high level control of the military unit group for use by the strategy manager. All basic military unit commands (attack, move, patrol, stand ground, etc) are made available to the strategy manager. More abstract commands, like attacking the opponent's base, are also made available. Keeping the military unit group on task is the responsibility of the second module. The final module removes slain units from the military unit group.

Recon Manager. As one of the managers that has just enough functionality to properly interface with the other managers, the recon manager uses aggregate unit information (number of military and worker units per player) and perfect information. The tactical and strategy managers request aggregate information from the recon manager. All academic and commercial RTS AIs currently make use of perfect information. As we develop the recon manager, we will remove the assumption of perfect information so that the recon manager must manage units in reconnaissance tasks.

Manager Interdependence

In this section we describe the relationship between the managers, and how the individual manager competencies are integrated to play a complete game. We demonstrate this through thought experiments where we examine the effects of removing individual managers.

If the income manager was removed, the agent would have no income and the production manager would be limited to using only the initial set of resources. Furthermore, there would be no method for the production manager to acquire a worker for construction or repair tasks. Under the default Wargus starting conditions, the agent would only be able to build four workers (which would sit idly by the town hall) and do nothing else.

If the production manager was removed, the agent would have no facilities to produce new units or buildings. This would result in the agent only being able to use the initial worker to either gather gold or lumber for the length of the game.

By removing the recon manager, the agent would have no way to sense its opponents or its own units, resources, and buildings, as well as have no knowledge of the map. The agent would not be able to perform any game actions.

With no tactics manager, the agent would not be able to coordinate multiple units for attack or defense. All economic and production tasks would proceed normally, but the agent could never attack; it would keep all units at its starting location, and engage in uncoordinated defensive battles when opponents attack (using the Wargus built-in default defense behaviors) until it is defeated. The best outcome one could hope for is a draw, brought on by the opponent running out of resources.

The removal of the strategy manager would result in agent capabilities similar to the removal of the production manager. Due to the strategy manager making all of the production and attack requests, the only actions the agent could perform would be the initial worker being told to gather resources by the income manager. The tactics manager could be used without the strategy manager to manage individual battles, but only given that initial battle conditions are set at the start of the simulation.

Results

The above section describes how the discrete competencies of the agent integrate to support the ability to play a complete RTS game. In this section we provide empirical results that the integrated agent performs well against two benchmark scripted AI opponents: the soldier's rush, and the knight's rush. Our agent outperforms the best reported results against these scripts. A soldier's rush involves staying at technology tier one, building a large collection of soldiers (weakest melee unit) and rushing the enemy base, presumably before they have had time to build a significant defensive force. The knight's rush involves building to technology tier two, building a number of knights, and attacking the enemy. Each script was tested on two maps. The first map was of a medium size (96 by 96 tiles) while the other was of a large size (128 by 128 tiles). Both maps had clear land routes between our agent's base and that of the scripted opponent. Each map and opponent combination was played 15 times.

As seen in Table 1, our agent bests the soldiers' rush most of the time on both maps. From analysis of the games, the lower win rate on the medium map was due to having less time to prepare for the early and vicious attack characteristic of the soldier's rush. The large map afforded the agent more time to build military capacity which resulted in better defense from the rush.

	Medium Map	Large Map	Both Maps
Soldier Rush	73%	86%	80%
Knight's Rush	60%	46%	53%

Table 1 – The percentage of games our agent won against each scripted opponent on each map.

By winning over half of the games played against the knights' rush, our agent has performed quite well in comparison to other Wargus agents. The knight's rush is considered a "gold standard" of a difficult scripted opponent. Other agents have defeated the knight's rush script, but have done so less frequently; our agent won 53% of the games against the knight's rush, while the highest reported win rate in the literature is 13% (Ponsen et al. 2005). Ponsen, in fact, hypothesizes that the knight's rush is an optimal Wargus strategy, as an explanation for why evolutionary learning plus dynamic scripting fairs poorly against the knight's rush. Expert human players, on the other hand, consider the knight's rush a fairly weak strategy, that they would only employ against a player with insufficient reconnaissance. If you know a player is performing a knight's rush, the counter-strategy is to perform a soldiers rush; while the enemy is spending all their economy on teching up to tier two, so they can produce knights, they are not building defensive forces, making them vulnerable to a soldier's rush. This is in fact what our strategy manager does, if it recognizes the enemy as performing a knight's rush.

Many of the losses suffered by our agent were due to the lack of sophistication of the tactics manager. Specifically, the tactics manager fails to concentrate military units in an area in either offensive or defensive situations. When many parallel decisions are being made elsewhere in the agent, small delays can be introduced in sending tactical commands to individual units, causing units to trickle towards engagements and be easily defeated. Future work in the tactics manager will focus on explicit formation management.

Conclusion

In this paper we have demonstrated an integrated agent capable of playing a complete RTS game. In contrast to other RTS agents in the literature, which have tended to employ an individual component to address a single aspect of the game (e.g. reinforcement learning for tactical management of small battles) or an individual component applied uniformly across the whole game (e.g. case-based retrieval of actions), our RTS agent employs multiple managers, where the needed managers were identified by a knowledge-level analysis of human expert play; the managers are integrated using the ABL reactive planner.

As we continue to develop our agent, the integrated modular architecture will allow us to strengthen individual competencies, and experiment with different techniques within different modules, while easily being able to test the effect on the global behavior of the agent. For example, in current work, we are integrating an HTN planner into the strategy manager to construct build orders, and adding behaviors to the tactics manager for formation management and dancing.

References

Bates, J.; Loyall, A.B.; and Reilly, W.S. 1992 Integrating Reactivity, Goals, and Emotion in a Broad Agent. *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, Bloomington, Indiana.

Buro, M. 2003. Real-time strategy games: A new AI research challenge. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) 2003*, pages 1534–1535. Morgan Kaufmann.

Chung, M.; Buro, M.; and Schaeffer, J. 2005. Monte Carlo Planning in RTS Games, In Proceedings of the IEEE Symposium on Computer Intelligence and Games, Colchester, UK.

Georgeff, M.P. and Lansky, A.L. 1987. Reactive reasoning and planning. In *Proceedings of the Sixth National Conference on Artificial Intelligence (AAAI-87)*, pages 677-682, Seattle, WA.

Guestrin, C.; Koller, D.; Gearhart, C.; and Kanodia, N. 2003. Generalizing plans to new environments in relational MDPs. In International Joint Conference on Artificial Intelligence (IJCAI-03).

Kovarsky, A. and Buro, M. 2006. A First Look at Build-Order Optimization in Real-Time Strategy Games, Proceedings of the GameOn Conference. 18-22. Braunschweig, Germany.

Mateas, M. and Stern, A. 2002. A behavior language for story-based believable agents. *IEEE Intelligent Systems*, 17(4), 39-47.

Mateas, M. and Stern, A. 2003. Integrating plot, character and natural language processing in the interactive drama Façade. In *Proceedings of the Technologies for Interactive Digital Storytelling and Entertainment* (TIDSE), Darmstadt, Germany.

Ontañón, S.; Mishra, K.; Sugandh, N.; and Ram, A. 2007. Case-Based Planning and Execution for Real-Time Strategy Games, Seventh International Conference on Case-Based Reasoning.

Ponsen, M.J.V.; Muñoz-Avila, H.; Spronk, P.; and Aha, D.W. 2005. Automatically Acquiring Domain Knowledge For Adaptive Game AI Using Evolutionary Learning. Proceedings of the Seventeenth Innovative Applications of Artificial Intelligence Conference. AAAI Press.

Ponsen, M.J.V.; Muñoz-Avila, H.; Spronck, P.; and Aha, D.W. 2006. Automatically Generating Game Tactics via Evolutionary Learning. AI Magazine, Vol 27, pp. 75-84.

Tambe, M. 1997. Towards Flexible Teamwork. *Journal of Artificial Intelligence Research* (7) 83-124.

Wintermute, S.; Xu, J.; and Laird, J.E. 2007. SORTS: A Human-Level Approach to Real-Time Strategy AI. Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference, Stanford, California.