Reactive Planning for Micromanagement in RTS Games

Ben Weber

University of California, Santa Cruz
Department of Computer Science
Santa Cruz, CA 95064
bweber@soe.ucsc.edu

Abstract
This paper presents an agent for commanding individual units in a real-time strategy game (RTS). The agent is implemented in the reactive planning language ABL and uses micromanagement techniques to gain a tactical advantage over opponents. Two strategies are explored focusing on harassment and unit formations. The agent is compared against the built-in AI of Wargus. The results show that reactive planning is a suitable technique for specifying low-level unit commands. Improving unit behavior in an RTS provides more challenging non-playable characters are partially alleviates players from the need to control individual units.

Introduction
The genre of video games known as real-time strategy is becoming increasingly popular. This is demonstrated by the fact that there are now international events held and broadcast to the entire world, such as Star Invitational (http://gsi.gomtv.com). One of the most interesting aspects of these events is the skillful command of individual units exhibited by professional players. By carefully controlling each unit, it is often possible for players to defeat outmatched armies. The managing of individual units is known as micromanagement and is one of the main aspects of RTS gameplay. One of the challenges faced by RTS developers is creating non-playable characters that utilize micromanagement strategies.

RTS players must balance a tradeoff between strategic and tactical levels of play. The strategic level of gameplay refers to long-term planning, such as maintaining an economy, producing combat units, and researching upgrades. The tactical level of gameplay refers to commanding units engaged in combat. As a player’s army grows in size, micromanagement becomes more difficult due to the number of units that must be individually controlled by the player. Therefore, novice players are forced to choose between strategic and tactical levels of play. Improving the low-level behavior of units will partially alleviate players from the need to micromanage units.

Traditionally, RTS games offer players commands, such as move, hold and attack. The attack command is commonly implemented as a hard-coded heuristic for selecting the next unit to attack. This approach leads to an inefficient utilization of the player’s army, due to a lack of collaboration between units. Therefore, the player is forced to manually select targets for individual units. The main problem with this approach is the hard-coded nature of the low-level AI. Many RTS engines expose an AI interface using a scripting language, such as Lua (Ierusalimschy et al. 2005). Scripting of high-level AI allows for specification of the strategic level of gameplay, but not the tactical level of gameplay.

There are several advantages to exposing a low-level AI interface for RTS games. The first benefit is customizable specification of individual unit behavior. This would enable players to script low-level behaviors to fit their needs. The second benefit is the ability to incorporate micromanagement strategies that were not anticipated by the game developer. Additionally, exposing low-level AI interfaces may enable new types of human-computer collaboration for RTS games. An RTS engine exposing low-level and high-level AI interfaces is the Open Real Time Strategy engine (Buro 2002).

Related Work
AI research in the domain of RTS has focused on high-level strategies for non-playable characters (Buro 2003, Walther 2006). Current research has focused on case-based reasoning (Aha et al. 2005), Markov decision processes (Guestrin et al. 2003) and reactive planning (McCoy 2008). However, Kovarsky and Buro (2006) state that the improvement of lower-level AI modules is important, because without effective solutions in this area, research on higher-level reasoning and planning cannot proceed. Therefore, research has branched in several directions, such as build order optimization and micromanagement.

Kovarsky and Buro propose a heuristic-based search for micromanagement of units (2005). They model the problem of micromanagement as an adversarial search and use randomization to limit the search space. The search uses several assumptions: all units have the ability to attack any opponent unit at any time and units are static objects, they cannot move. The results demonstrate that collaboration between units increases the chances for victory. However, the assumptions required for the search do not hold in RTS games. Additionally, the search
represents a fixed strategy, based on the encoding of the objective value.

Another approach is the use of Markov decision processes for micromanagement of units (Guestrin et al. 2003). Guestrin et al. utilize relational Markov decision processes to create a strategy for three versus three melee unit battles. The results demonstrate fairly complex behaviors, such as focus firing and switching targets as the initial target becomes severely injured. The approach scales to four versus four unit battles, but is not practical for larger matches.

Recent RTS games have provided more options for the behavior of groups of units. For instance, Warcraft3 provides the ability to force units to move as a group with the melee units leading the group. While this option enforces a good formation of units while moving, the attack behavior is based on a hard-coded heuristic. Therefore, the player is still required to micromanage individual units in order to effectively utilize an army.

**Micromanagement**

Micromanagement includes a class of strategies in RTS games. These strategies include unit formations, unit targeting, dancing and harassment. Unit formation refers to how units position themselves relative to other units in the group. Typically, it is desirable to have melee units in the front of the group, because melee units have a short range and can take more damage. Unit formations can also use the topology of the map to gain an advantage. For instance, a player can utilize a chokepoint in the map as a defensive position.

Unit targeting is the process of selecting which units to attack. Generally it is more efficient to kill off units one at a time, then to disperse damage evenly across a group of units. However, it is not usually beneficial to attack single units with an entire army, due to range and movement constraints. Therefore, it can be difficult to find the optimum number of units to attack simultaneously. Unit targeting also refers to the order in which units are targeted based on type. For instance, a player may wish to kill all enemy melee units before engaging enemy siege units.

Dancing is the process of moving individual units in order to gain an advantage. Dancing involves moving injured units out of the range of enemy units in order to recover or heal. Often, dancing causes an enemy unit to revert to its higher-level command and acquire a new target. The dancing unit can then re-target the enemy unit without taking damage. Conversely, if the enemy unit does not revert to its higher-level command, it will follow the dancing unit. This enables other units to target the enemy unit, which is now chasing the dancing unit rather than attacking. This form of dancing is known as kiting.

Micromanagement strategies can also be used to harass enemy units. For instance, a fast, range unit can harass a slow, melee unit. The range unit can repeatedly attack a melee unit and run away until the melee unit is defeated.

Harassment enables a player to kill enemy units while incurring minimal damage, but requires a large amount of the player’s attention to be focused on a small aspect of the game.

**Framework**

The ABL/Wargus framework was utilized to implement low-level AI behavior for units in an RTS game. The framework provides a layer of Java code that translates raw Wargus game state to ABL sensor data and from ABL primitive acts to Wargus game actions (McCoy 2008).

**ABL**

ABL (A Behavior Language) (Mateas and Stern 2002) is a reactive planning language for authoring sophisticated game AI. It is an extension of the agent language Hap and does not require a commitment to a formal domain model. An agent in ABL pursues parallel goals, which are satisfied by sequential and parallel behaviors. ABL was originally designed to support the creation of autonomous believable characters. However, ABL has also been utilized for RTS AI (McCoy 2008).

**Wargus**

Wargus is an open-source clone of Warcraft II that utilizes the Stratagus game engine (Ponsen et al. 2005). Low-level commands for units are hard-coded in the game engine. The attack command is implemented such that a unit always attacks the enemy with the best objective value. The attack objective value is based on whether the unit is in range, whether the enemy unit can attack back, the remaining health points of the unit, and distance to the unit.

Wargus was modified to overwrite the default low-level behaviors of units. The attack heuristic for ABL controlled units was disabled, forcing the planner to decide which units to attack. The fleeing behavior was disabled, delegating fleeing strategies to the planner and providing a mechanism for forcing units to stay in formation. Additionally, Wargus was modified to allow units to move during the cooldown period after attacking. Without this modification, harassment strategies are not feasible.

A subset of unit types was selected from the types available in Wargus. To allow for even matchups, the human race is selected for both players. Armies consist of footmen (melee), archers (range) and gryphon riders (air). Heroes and casters were not considered, due to the additional complexity of spells. This subset provides a rich enough selection of units to compare reactive planning against current techniques.

**Implementation**

Two ABL agents were implemented focusing on different aspects of micromanagement. The first agent utilizes a hit and run strategy to harass enemy units. The second agent...
uses unit formations, unit targeting, and dancing techniques to gain a tactical advantage.

Implementing unit behaviors in ABL requires defining predicates, actions and behaviors for the domain. Actions in the Wargus domain include commands such as attack, move, follow and stop. These actions are included in the ABL/Wargus framework. It was necessary to add additional predicates in order to specify low-level behavior. Several spatial predicates were added to the domain to enable reasoning about formations and targeting. These predicates include x and y coordinates, distance to the nearest enemy, direction of the nearest enemy, and an adjacency matrix. The adjacency matrix contains information about the presence of allied and enemy units in adjacent squares (see Figure 1).

The ABL agents command several units in parallel. A flag was added to units to limit the number of commands that can be issued to a single unit in a given time period. Any command that triggers an action in Wargus results in setting a unit’s busy flag. This flag is used for preconditions in behaviors that trigger actions in Wargus. A timer is used to clear the busy flags twice a second. Therefore, each unit is limited to at most two Wargus commands a second. This modification was necessary to successfully interleave planning and execution.

Two behaviors in the harassment agent satisfy the move goal. The first behavior checks that no units are currently engaged and moves the gryphon rider to the nearest enemy unit. The second behavior checks for an enemy unit within range and sets the engaged flag when the condition is met.

The harass move goal is accomplished by four behaviors. The preconditions of these behaviors require that the engaged flag is set and check that the attack timer is below a threshold value. The four behaviors correspond to the direction of the nearest enemy unit, with respect to the harassing unit. For example, if the enemy unit is to the north, the harassing unit will move west of the enemy unit. The combination of these behaviors causes the harassing unit to move clockwise around the enemy unit in a diamond-like pattern. This movement pattern allows the harassing unit to move constantly, while the enemy unit remains stationary. Additionally, each harass move behavior increments the value of the attack timer.

The attack enemy goal is satisfied by a single behavior. The attack behavior verifies that the engaged flag is set and checks that the attack timer is equal to a threshold value. When these preconditions are met, the harassing unit attacks the enemy unit. After attacking, the attack timer is set to zero.

The behaviors for the harassment agent are implemented such that only one goal can be accomplished at any given instant. This is achieved through the use of the engaged flag and an attack timer in the preconditions. The design of the agent reflects sequential goals due to the procedural nature of harassment.

Harassment Agent

The harassment agent utilizes a hit and run technique in order to defeat enemy units in gryphon rider battles. The agent exploits the fact that it is possible to dodge enemy projectiles by constantly moving, as shown in Figure 2. After attacking, an enemy unit begins the cooldown phase and is vulnerable to attack. During this period, the harassing unit returns fire and then continues the movement pattern. The harassing unit must move in a pattern that limits the movement of the enemy unit, to prevent the enemy unit from also dodging projectiles. The agent uses a circular movement strategy to achieve the desired behavior. The agent implements this strategy by continuously pursuing three parallel goals: move, harass move and attack enemy.

Formation Agent

The formation agent utilizes unit formations, unit targeting and dancing strategies for medium-sized battles. The agent first builds unit formations. This is accomplished by pursing the assign formation leader and build formation goals in parallel. Next, the agent moves the formation towards enemy units. This behavior is achieved by the
move goal. Finally, the agent attacks engaged enemy units by pursing the attack and dance goals in parallel.

**Formation Building**
The assign formation leader and build formation goals are used to create unit formations in Wargus. The formation leader is a unit selected by the planner used for iteratively building a formation. The assign formation leader goal can be accomplished by two behaviors. The first behavior selects a melee unit as the formation leader and the second behavior selects a range unit as the formation leader. The first behavior has a higher specificity, because a melee unit is preferred for the formation leader.

The build formation goal is satisfied by several behaviors, which are specific to melee and range units. The behaviors for melee units are as follows:
- Move to the left of the formation leader
- Move down until there is an opening to the right
- Move the unit into formation
- Command the unit to hold position
- Set the unit’s inFormation flag

There are additional behaviors to deal with contingencies that arise from the wayfinding algorithm in Wargus. The behaviors for range units are similar, except range units line up to the right of the formation leader, rather than the left. The resulting formation is shown in Figure 3.

**Formation Movement**
The move goal is accomplished by two behaviors. Both behaviors require that all of the unit’s inFormation flags are set. The first behavior checks that the grid location to the left is open and then moves there. The behavior also enforces units to not get more than one grid location in front of the formation. The second behavior checks if there is an allied unit in the grid location to the left and commands the unit follow the adjacent unit. The combination of these behaviors causes the units to move forward while maintaining the formation structure.

**Formation-Based Attacks**
The formation agent attacks enemy units by pursing the attack and dance goals. There are several behaviors for achieving the attack goal. The behavior to use is dependent on the remaining allied and enemy units. The preference of attack behaviors is as follows:
- Attack an enemy unit if two or more allied units are attacking it and the unit is in range
- Attack an enemy unit if one or more allied units are attacking it and the unit is in range
- Attack the weakest melee unit in range
- Attack the closest melee unit
- Attack the weakest range unit in range
- Attack the closest range unit.

The specificities of the behaviors cause the units to attack all melee units before engaging range units. Also, a unit will only change targets when preconditions for a behavior with a higher specificity are met.

The attack behaviors result in focused fire, because group attacking is preferred over individual attacks. There is no flank attack behavior, but this is achieved by the attack closest unit behaviors. The specification of attack behaviors is more complex than the heuristic used by Wargus, which does not consider group attacking.

The formation agent implements dancing strategies by moving units out of formation. Three behaviors achieve the dancing goal. The first behavior checks if the health of a melee unit is below a threshold and moves the unit out of combat when the condition is met. The second behavior is the same as the first behavior but monitors the health of range units. The third behavior updates the dance timer of units and specifies when they should return to battle.

**Formation-Based Attacks**
The formation agent attacks enemy units by pursing the attack and dance goals. There are several behaviors for achieving the attack goal. The behavior to use is dependent on the remaining allied and enemy units. The preference of attack behaviors is as follows:
- Attack an enemy unit if two or more allied units are attacking it and the unit is in range
- Attack an enemy unit if one or more allied units are attacking it and the unit is in range
- Attack the weakest melee unit in range
- Attack the closest melee unit
- Attack the weakest range unit in range
- Attack the closest range unit.

The specificities of the behaviors cause the units to attack all melee units before engaging range units. Also, a unit will only change targets when preconditions for a behavior with a higher specificity are met.

The attack behaviors result in focused fire, because group attacking is preferred over individual attacks. There is no flank attack behavior, but this is achieved by the attack closest unit behaviors. The specification of attack behaviors is more complex than the heuristic used by Wargus, which does not consider group attacking.

The formation agent implements dancing strategies by moving units out of formation. Three behaviors achieve the dancing goal. The first behavior checks if the health of a melee unit is below a threshold and moves the unit out of combat when the condition is met. The second behavior is the same as the first behavior but monitors the health of range units. The third behavior updates the dance timer of units and specifies when they should return to battle.

**Formation-Based Attacks**
The formation agent attacks enemy units by pursing the attack and dance goals. There are several behaviors for achieving the attack goal. The behavior to use is dependent on the remaining allied and enemy units. The preference of attack behaviors is as follows:
- Attack an enemy unit if two or more allied units are attacking it and the unit is in range
- Attack an enemy unit if one or more allied units are attacking it and the unit is in range
- Attack the weakest melee unit in range
- Attack the closest melee unit
- Attack the weakest range unit in range
- Attack the closest range unit.

The specificities of the behaviors cause the units to attack all melee units before engaging range units. Also, a unit will only change targets when preconditions for a behavior with a higher specificity are met.

The attack behaviors result in focused fire, because group attacking is preferred over individual attacks. There is no flank attack behavior, but this is achieved by the attack closest unit behaviors. The specification of attack behaviors is more complex than the heuristic used by Wargus, which does not consider group attacking.

The formation agent implements dancing strategies by moving units out of formation. Three behaviors achieve the dancing goal. The first behavior checks if the health of a melee unit is below a threshold and moves the unit out of combat when the condition is met. The second behavior is the same as the first behavior but monitors the health of range units. The third behavior updates the dance timer of units and specifies when they should return to battle.
Results

The ABL agents were compared against the built in AI of Wargus in several scenario configurations. The harassment agent was tested in a scenario in which ABL commands a single air unit and the enemy commands two air units. The two air units were far apart on the map and were engaged separately by the harassment agent.

The formation agent was compared against the default attack move behavior in Wargus. The enemy units were hard coded to attack move to the initial position of the agent’s units. The formation agent was compared against several unit configurations and the results are shown in Table 1. Each scenario configuration was executed 5 times. The 10 versus 10 units in formation scenario consisted of each player having 5 melee units and 5 range units in the formation shown in Figure 3. The 10 versus 10 units not in formation scenario consisted of each player having 5 melee and 5 range units with the initial formation shown in Figure 5.

<table>
<thead>
<tr>
<th>Win Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 vs. 2 air units</td>
</tr>
<tr>
<td>3 vs. 3 melee units</td>
</tr>
<tr>
<td>5 vs. 5 melee units</td>
</tr>
<tr>
<td>5 vs. 5 range units</td>
</tr>
<tr>
<td>10 vs. 10 units in formation</td>
</tr>
<tr>
<td>10 vs. 10 units not in formation</td>
</tr>
</tbody>
</table>

Table 1 – Win ratios against the built in AI of Wargus

The harassment agent was successful against the default AI of Wargus during each execution of the harassment scenario. The scenario demonstrates that reactive planning is capable of implementing low-level procedural behaviors.

The formation agent had varied success in melee versus melee battles. As the number of enemy units increased, the win ratio of the agent decreased. In small melee battles, the dancing behavior resulted in kiting. Kiting makes the enemy units vulnerable to attack while chasing a dancing unit. This behavior is shown in Figure 4, where the enemy units with identifiers one, two and three are chasing the unit with identifier six.

The agent failed to win a majority of battles in five versus five melee unit scenarios. The wayfinding algorithm in Wargus often causes units to cross paths when attacking. When this behavior occurs, the formation breaks apart and the agent usually loses. The formation agent typically won when units remained in formation after receiving an attack command.

In range versus range unit battles, the formation agent performed comparably with the built in AI of Wargus. The units controlled by ABL immediately focus fire on a single enemy unit. The units controlled by Wargus first target individual units and then focus fire on the weakest unit. The unit targeted by the enemy units then attempts to dance in order to kite the enemy units. The formation agent won battles in which the dancing unit successfully kited enemy units. If the dancing unit died before getting out of enemy range, then the formation agent lost. The formation agent would potentially have a higher win ratio on this scenario if the reaction times of units were shorter than half a second.

The formation agent was unsuccessful at defeating the default AI in formation-based battles. There were several reasons to explain this result. The formation structure starts to dissolve as units begin focus firing, because units need to get in range of enemy units. Additionally, the wayfinding algorithm often caused units to take abnormal paths to attack units, further breaking down the formation. Dancing was not effective for melee units, because range units were in the way. Also, commanding melee units to dance reduces the damage output of the units.

The formation agent was required to attack all melee units before engaged range units. This constraint often worked against the agent when melee units needed to traverse several grid locations to attack a unit, rather than engage adjacent enemy units. The built in AI of Wargus did not utilize this constraint and often gained a significant advantage by attacking range units early.

Conclusion

This paper has demonstrated the use of reactive planning for implementing low-level behaviors of units in Wargus. Two agents were implemented in ABL focusing on harassment and formation strategies. The results show that
reactive planning is well suited for procedural harassment techniques. However, the results for the formation-based scenarios show that reactive planning may not scale well to large battles. Overall, formations did not improve the performance of the agents. Also, micromanagement strategies such as dancing and focus fire improved performance for small battles, but were not as successful for larger battles.

The built-in AI of Wargus performed surprisingly well against the complex attack rules of the formation agent. Behaviors such as focus fire emerge from the formulation of the attack objective value. However, using an objective function requires the introduction of free variables to the attack rule. Reactive planning provides a formal method for specifying attack behavior, but has yet to outperform heuristics.

Successfully commanding a large number of units in an RTS requires short reaction times. The ABL agents presented planned at a low-level and limited the reaction times of units to half a second. Future work will explore the use of abstracting the reactive planner from low-level details and focus more on operational and tactical levels of gameplay. However, this direction of research still requires the specification of unit behavior. A potential approach is the use of programmable finite state machines to implement low-level behavior. The Open Real Time Strategy engine provides a framework for implementing this approach. Additionally, future work will consider larger battles and utilizing additional unit types.

References


McCoy, J. 2008 An Integrated Agent for Playing Real-Time Strategy Games. Submitted to AAAI.
