# An Effective and Efficient Real Time Strategy Agent

**Kurt Weissgerber, Brett J. Borghetti** and **Gilbert L. Peterson**
Air Force Institute of Technology*
Graduate School of Engineering
2950 Hobson Way
Wright Patterson AFB, OH 45433-7765, USA
kurt.weissgerber; brett.borghetti; gilbert.peterson@afit.edu

### Abstract

Real Time Strategy (RTS) games present a challenge for the field of artificial intelligence. The size of the state space often makes machine learning solutions intractable and as a result, developers turn towards hand-tailored solutions such as scripts. In this work, we present the Killer Bee Artificial Intelligence (KBAI) agent which uses a predictive nearest neighbor classifier to make decisions in real time. KBAI has a no-loss record against four hand tailored scripts in the RTS game Spring.

## Introduction & Related Work

Several learning agents have been developed for the RTS domain. These agents have been successful across various platforms: WARGUS (Spronck et al. 2006), Bos Wars (Kok 2008), Spring (Bakkes, Spronck, and van den Herik 2008) and ORTS (Chung, Buro, and Schaeffer 2005). All of these agents were able to learn winning strategies against scripted opponents after a number of learning iterations, ranging from eight to sixty games. However, according to the respective authors, all were weak when pitted against rush opponents.

The Killer Bee Artificial Intelligence (KBAI) agent determines the important characteristics of an opponent by examining game traces, and then uses these important characteristics to make a compact representation of the state. Using the nearest neighbor classifier presented in (Weissgerber et al. 2009), which performs both feature reduction and sample generalization on game traces, KBAI determines which values from a small set of key features lead to the opponent's success and which lead to its defeat and links these features to actions. The appropriate action is selected through observation of the current state and its proximity to the samples in the classifier. KBAI agents are both effective and efficient: they have a zero loss record when tested against four different scripted agents and win faster than other dynamic agents.

*The views expressed in this article are those of the authors and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.



## KBAI Development

KBAI requires a compact state representation. The state which KBAI uses is a vector of features $F$, where each feature is identified by the unit or building it pertains to, and the value of each feature is the difference between the number the agent owns minus the number the opponent owns. To make decisions about what actions to pursue, KBAI uses a nearest neighbor classifier that differentiates winning states from losing states. To reduce memory requirements, the classifier maps the values of the features of $F$ to a set of winning centers $W$ and losing centers $L$. A center is an exemplar of a winning or losing state in the domain.

The appropriate action to take is determined by picking a feature and attempting to increase its value. When choosing which feature to influence, the agent is trying to get from the current state $S$ to some future state $S'$. All the possible $S'$ candidate states are computed by determining the effect of focusing on a particular feature. While the $S'$ values can not be computed with complete certainty, they can be estimated by specifying some constant increase amount for each feature. One divided by the highest observed value for a feature is used as the increase amount when computing $S'$ values.

To pick the best next state, some evaluation function must be used to determine the expected value of each $S'$. The centers give a natural way to do this: the candidate state which is closest to a center in $W$ is the best possible next state. Distance from a candidate state to a winning center is computed with a Euclidean distance function. The agent determines the candidate state which is closest to a center in $W$ and determines the feature in $S'$ which is furthest below the target values. This becomes the "active feature". If all the target values are less than the values of the current state, the feature with the smallest difference is selected.

Many human players will confirm the most important aspect at the beginning of an RTS game is to build units as quickly as possible. To accomplish this, KBAI agent's focuses on unit construction right from the beginning. The agent determines the type of unit to produce at game start by examining the features in $F$. The agent looks through the state representation for features which require heavy tanks. If no heavy tank features are in $F$, then the agent looks for features which require heavy infantry, then light tanks, and finally light infantry units. The agent produces as many of the appropriate unit production buildings as possible, along

with required infrastructure. Any production buildings completed produce the chosen production unit.

When enough units have been produced, the agent selects a target. KBAI agents attack the opponent's "centers of gravity": the units/buildings which have a large impact on the opposing strategy being pursued. There are two different targeting scenarios faced by KBAI agent's: when some of the opponent's units can be seen and when they cannot. In the former case, a target is selected from the units which are visible. Offensive units are targeted first, then buildings. In the latter case KBAI uses perfect information to get a list of all of the enemy's units. This list is searched for *strategic targets*. Strategic targets are based on the active feature: if the active feature is an infantry unit, the strategic target is infantry unit production buildings. If the active feature is metal production, then the strategic target is metal production buildings. The closest strategic target is computed by determining the center of mass (average of the Euclidean locations) for all of the available offensive units.

## Experiment

Our goal is to build an agent which is effective and efficient against many different opponents. The effectiveness of an agent is its win percentage. Efficiency is an agent's time-to-win. An effective, efficient agent is one which wins fast.

KBAI is tested in the RTS game Spring, an open source RTS platform. Each participant in a Spring game starts with a Commander unit. Games end when one participants Commander is destroyed. To validate the effectiveness and efficiency of the dynamic agent KBAI, its performance is compared to that of a dynamic random agent (DRA) and two full featured agents developed for the Spring domain, RAI and KAI (available with the Spring distribution). Each agent is tested against four scripted agents in a fifty game match.

KBAI agents are only allowed to produce the units which are present in the scripted opponents: medium/light tanks and heavy/light infantry units. It builds in random locations and has no algorithm to keep the commander protected. KBAI generates a different nearest neighbor classifier for each script from twenty-four game traces, subject to the constraints $|F| \leq 6$, $|W| \leq 6$ and $|L| \leq 6$. Forty five different features are available. All centers were vectors of real values on the interval $[-1, 1]$. DRA is an agent which uses the same underlying architecture as KBAI, but makes a random choice whenever KBAI would reason on the current state.

Four different scripted agents were developed for use as opponents. The **infantry rush** and **tank rush** quickly produce and attack with cheap units. The **blitz** creates an unstoppable army to overwhelm the opponent. **Turtle** builds a "shell" of defensive buildings and waits for the opponent to attack, responding with a large counteroffensive.

## Results, Analysis & Future Work

Table 1 shows the winning percentage for all agents versus the scripted agents. KBAI clearly outperforms all other dynamic agents, winning 100% of the time against all the scripted agents. The only other effective dynamic agent is DRA, which uses the same framework as KBAI: it builds as many units as it can as fast as possible. As discussed earlier, this is generally the determining factor of an RTS game.

Table 1: Dynamic agent win rate against the scripted agents.

| Agent | Blitz | Infantry Rush | Tank Rush | Turtle |
|---|---|---|---|---|
| KBAI | 100% | 100% | 100% | 100% |
| DRA | 74% | 90% | 68% | 70% |
| KAI | 36% | 18% | 30% | 38% |
| RAI | 8% | 66% | 14% | 20% |

If an agent is effective, then we examine whether it is efficient. An effective agent is one which wins at least 60% of its games. Table 2 has the mean game length for the effective dynamic agents, calculated only from games which the dynamic agent won. KBAI has a lower mean game length against all opponents, and two-sample t-tests for a difference between the game lengths of KBAI and DRA give significant evidence of a difference.

Table 2: Mean time to win (seconds) for effective dynamic agents.

| Agent | Blitz | IR | TR | Turtle |
|---|---|---|---|---|
| KBAI | 507 | 540 | 512 | 548 |
| DRA | 604 | 750 | 724 | 905 |

KBAI outperforms the other test agents, in terms of both effectiveness and efficiency. It is a robust system which can be used to defeat different types of scripted agent, without the weakness to rush strategies of other reinforcement learning approaches.

Future areas of interest include expanding the action space for economic decisions: currently KBAI can only produce four different attack unit and six building types. We hope to increase this number and show the continued viability of our method, as well as seek ways to combat more dynamic agents.

## References


Bakkes, S.; Spronck, P.; and van den Herik, J. 2008. Rapid adaptation of video game ai. In *9th International Conference on Intelligent Games and Simulation GAME-ON*.

Chung, M.; Buro, M.; and Schaeffer, J. 2005. Monte carlo planning in rts games. In *IEEE Symposium on Computational Intelligence and Games*.

Kok, E. 2008. Adaptive reinforcement learning agents in rts games. Master's thesis, University Utrecht, The Netherlands.

Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2006. Adaptive game ai with dynamic scripting. *Machine Learning* Volume 63:pp 217–248.

Weissgerber, K.; Borghetti, B.; Lamont, G.; and Mendenhall, M. 2009. Towards automated feature selection in real-time strategy games. In *GAME-ON North America*, 25–32.