SCAIL: An integrated Starcraft AI System

Jay Young, Fran Smith, Christopher Atkinson, Ken Poyner and Tom Chothia

***Abstract*—We present the work on our integrated AI system SCAIL, which is capable of playing a full round of the Real-Time Strategy game Starcraft. Our system makes use of modern AI techniques such as particle filtering, on-line machine learning, drive-based motivation systems and artificial emotions, used to find novel structure in the dynamic playing environment, which is exploited by both high and low-level control systems. We employ a principled architecture, capable of expressing high level goal-directed behaviour. We provide an overview of our system, and a comparative evaluation against the in-game AIs of Starcraft, as well as thirteen third party systems. We go on to detail how the techniques and tools we introduce provide advantages to our system over the current state-of-the-art, resulting in improved performance when competing against those systems.**

I. Introduction

We are interested in building AI systems capable of acting in dynamic, complex domains. Such domains may exhibit the feature of being oversubscribed, by which we mean that there may many ways for an agent to accomplish mission-level goals, necessitating the management of various trade-offs in deciding which course of action to commit to. As the complexity of a domain grows, it becomes increasingly difficult to envision how a good solution might look, however we do know that any such solution will be an *integrated system*, gathering together a mixture of various AI techniques to address individual sub-problems.

Such an integrated system will be composed of a broad set of information processing systems and mechanisms for managing internal state and control, as well as affectors for altering the environment and executing plans. Our view is that we would prefer these systems to be general mechanisms for accomplishing tasks in the operating domain. That is, we would like to bestow *tools* upon AI systems, as opposed to full solutions encoded at design-time. An autonomous system should ideally then learn how its tools can be best utilised in order to produce novel solutions for problems it might encounter that can not be anticipated at design-time. In Starcraft the need for such capabilities is clear, as we, as designers, cannot pre-emptively anticipate every situation an AI system might find itself in. An ideal system should be able to creatively adapt, as expert human players do. However, much work still needs to be done before we can reach this point, as questions exist as to how we might process, structure and present information about Starcraft to an AI system, and how this might be exploited.

Crucial to the furthering of AI research in this area is the nurturing of a research community to maintain a body of work on integrated solutions and techniques. It is this body of work that we contribute to.

II. Starcraft

Starcraft is a Real-Time Strategy game released by Blizzard Entertainment in 1998[1]. The game requires a human player to engage in high-level goal-directed planning, reasoning under uncertainty, creative adaptation, and the management of limited attentional resources. This must be accomplished in real-time, which exasperates many of the already difficult AI challenges present. In recent years the game has become the focus of interest from the AI community, supported by the release of the Brood-War API (BWAPI[2]), a software API which allows for the injection of code into the Starcraft game engine, and facilitates the production of third-party AI players. This has led to several high-profile tournaments, such as those run by IEEE CIG and AIIDE, which pit these AIs against each other.

III. Related Work

Starcraft is currently the focus of a small, growing, research community, which seeks to construct systems to tackle the AI problems presented by the domain. This comprises a body of work employing techniques such as Bayesian programming [1], Neural Networks [2], Swarm Intelligence [3], [4], and work on systems that seek to learn from analysing replays of expert human players [5], [6]. There also exist tournaments, with major events run by IEEE and AIIDE, in which large, integrated systems compete against each other. Specifically in our own work, we are interested in the kind of integrated architectures and information processing mechanisms that support goal-directed behaviour [7], and allow systems to play a full round of the game for entry into such tournaments.

IV. Overview

Our primary contribution is a description of the architecture and techniques used in a system capable of playing a full game of the real-time strategy game Starcraft as the Protoss faction. We introduce a range of techniques that extend the current state-of-the-art in this domain by exposing information and interfaces that allow our system to make intelligent decisions at both high and low levels. Starcraft is as yet a young domain and test-bed for AI work, and so part of our contribution is to disseminate our ideas as a basis for future work, and as a furthering of existing knowledge. Due to space constraints our description will however remain relatively high-level.

Several elements of our work are informed or inspired by our pre-existing experience with mobile robotics, with one of our long-term research aims being to learn about

[1] http://us.blizzard.com/en-us/games/sc/

[2] http://code.google.com/p/bwapi/

978-1-4673-1194-6/12/$31.00 ©2012 IEEE

systems used for the control of mobile robots as applied to virtual domains. In this sense, our contribution extends to introducing techniques and vocabulary to the literature that are not prevalent in the current work, and so may provide interesting avenues for future work.

A. Particle Filters

Throughout this paper we make extensive use of the term *particle filter*, which is a technique that may be more specifically described as a form of stochastic optimisation [8]. In our system, we employ particle filters to reason about and locate structure in the virtual environment of Starcraft. We define a particle filter as consisting of a point p in space, around which we generate a Gaussian distribution of n child points. These points are then filtered through heuristic evaluation, leaving only those points that meet some specified criteria. The number and variance of points generated is configurable, and the exact nature of filtering via heuristic evaluation depends on the implementation context in which the filter is to be employed. For instance, a micro-management system for controlling a Dragoon might employ a particle filter during combat to locate nearby points in space that are in range of the fewest enemy units. This produces a set of points from the initial distribution that meet the heuristic criteria, which can be used or further filtered (i.e. Perhaps by finding the points closest to the unit's current position) to move the unit away from danger. Particle filters are computationally inexpensive, and are used in-real time in our system as tools to find solutions such spacial problems.

V. Macro Management

We split the discussion of our system into two sections: Macro and Micro. By the first, we mean those systems concerned with high-level strategy, such as build orders, unit production and strategic decision making. By the second, we refer to those systems concerned with low-level control of individual units, and groups of units, in the game. We begin by describing systems intended to address the problems of macro-level management.

A. Task-Based architecture

When developing Starcraft AI projects we would like to interface with the game in powerful ways, affording both high and low-level control. The base components of BWAPI go some way towards realising this goal, but provide only a basic layer of functionality. In order to produce sophisticated AI programs, we need to add-in custom functionality and architectures on top of this layer. How to do this is not immediately obvious, however, nor is it a trivial task. To counter the inexorable complexity such systems tend to take on, we advocate a principled and extensible architecture design.

We implement a goal generation and management architecture based around the idea of a unit of work we call a Task. A Task is a request to initiate some form of macro-level action, such as unit training, building construction, research, attacking, defending etc. Each Task encloses a reactive plan designed to accomplish a high-level goal, as well as tools required for monitoring its own execution and repairing failures as and when they occur. Tasks are generated by independent task generators – for example, a build order is given as an ordered sequence of construction Tasks. Similarly, to produce combat units, we use drives to periodically instantiate training Tasks, the mechanisms behind which will be discussed further on.

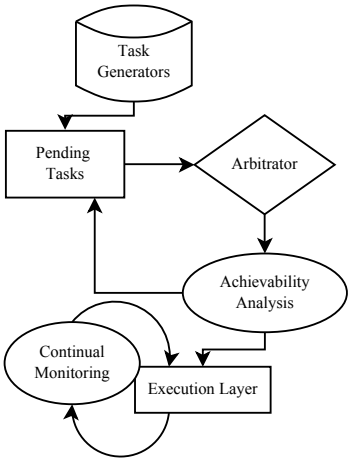

Fig. 1. High-level task system diagram

A Task related to constructing a building might be split down into:

- Evaluating whether or not the player has the required tech or resources.
- Finding a suitable worker to construct the building.
- Locating a suitable building location.
- Internally reserving resources for the Task.
- Monitoring the progress of construction.
- Alerting the system regarding completion or failure.

Since plans will be roughly identical across a given class of task – such as building or training – we generalise along these lines to create several task-types. Tasks are passed on to an arbitrator, which manages conflicts in situations where multiple tasks require access to the same resources. This overall workflow is broadly depicted in *Figure 1*.

We also express activities such as scouting, attacking and harassing the enemy using the task system. This is accomplished either through specifying certain tasks as part of an opening sequence, such as a build order, or generating them independently via systems that monitor changing game conditions and instantiate tasks as required to address them. For instance, we employ a Defense Monitor which instantiates tasks to rally idle combat units to the aid of friendly bases that are under attack. Another system monitors the number of workers at a base, and instantiates training tasks if the numbers fall below pre-set thresholds – the same system is capable of requisitioning workers from other bases if they are mined out or oversubscribed.

B. Belief Management

In Starcraft, a player is only able to observe the real-time state of the game environment directly surrounding their own units or buildings, which is known as the fog of war effect. In order to reason about entities it cannot immediately observe (but knows to exist), an AI must be equipped with a memory. We address this need by maintaining a database of beliefs about observations that have been made in the past, which may be simple facts such as The enemy has building T, located at (x,y), in such a case a belief will persist until the unit pertaining to it has been observed to have been destroyed, at which point it is abandoned. Beliefs might also include more complex processes, such as a belief of the strength of an opponent based on observed information.

Inference is performed via a rules-based engine. Particular units and buildings have networks of pre-requisites that must be constructed before they themselves can be trained or constructed – available through the BWAPI interface. Should the AI see such a unit, it uses these rules to infer that any pre-requisites must also exist, though it will not be aware of their physical location immediately. We therefore instantiate blank beliefs regarding these elements. Later on, as further observations are made (through scouting, for instance) the AI will fill in these blanks and ground abstract beliefs in concrete terms.

Through this belief-based approach we are able to produce a model of the state of an opponent, which can be utilised in order to inform strategic decisions such as which units to employ against an opponent and where weak-spots in defences may be. The database is also accessible by all units at the micromanagement level, meaning that units possess a shared, global view of the game state. This affords inter-unit collaboration, for example by allowing scouts to spot targets for offensive forces, and also renders the ”Blind” effect mostly ineffective, as so long one of our units still has full sight, their vision is shared through the belief database with all others.

As a tool to work with this space, we implement a simplified version of the DBSCAN algorithm [9], which operates over our belief space and clusters units based on spacial density. This provides the ability to observe a unit and immediately and efficiently retrieve a list of nearby units that may be supporting it, one use of which is for force calculations. This also lets us partition units off into groups, and so we can also reason about which groups of units exist, and which are supporting each other. If these units are not currently visible, then their last believed locations are reported. This lets the system make more reliable decisions during protracted combat over large distances or differing terrain heights, where units might flow in and out of line of sight. The clustering technique can also be used offensively, for instance, by applying the algorithm to a model of an enemy base to find clusters of valuable buildings and units upon which to launch a nuclear strike.

VI. Attack Drive

A difficult question in RTS games, both for human and AI players, is that of when a player should launch an attack. A human player would likely be armed with some knowledge of the build order and timing of the opponent, and through this might be able to anticipate weaknesses and when they should attack in order to take advantage. How we might go about bestowing this ability on an AI system is a highly complex and difficult question. One of the tools for answering this question is our belief database, which allows us to talk about what we expect state of the opponent might be. But the flow of information in this dimension is discontinuous - we are only capable of observing sparse snap-shots of part of the state of the opponent. While we may believe that the opponent has a force strength of n based on our previous observations, in-between observations the value of those beliefs degrades (assuming that the opponent is continually expanding their force, at some unobserved rate). There is then a finite time horizon on the usefulness of a belief. This then requires a system to keep its knowledge about an opponent up-to-date, typically through scouting, but this may not always be possible – especially late in the game – as a base may be heavily defended and impenetrable to scouts.

Fig. 2. System producing clusters of three perceived groups of units. Members of each group are coloured red, green and white respectively.

We think about the problem in two dimensions. First, we wish to field a force that we have some justifiable belief will be able to pose a threat to that possessed by the enemy. Second, we accept that there is time pressure at play in making such decisions, preferring to make them based on information that has been recently acquired.

The central tool we employ is a comparison of the number of minerals spent on the enemy force as compared to our own, including upgrades[3].

We use the disparity between friendly and enemy mineral expenditure on forces as a drive, which motivates the system to act [10], [11]. In humans, an example of a similar drive is hunger – our hunger level rises steadily until we eat, at which point it is satisfied for a time. We utilise the same mechanism here.

The system launches attacks on bases when it believes it

[3]It is certainly possible to think of other comparators that might employ richer representations.

can field a force that is worth more than that defending, and we employ a threshold on the drive to trigger attacks. We initially set this threshold at 10% (i.e. the believed enemy force must be worth at least 10% less than the force we are capable of fielding), which we ramp up to 30% over time, so as to not dissuade the system from launching early-game attacks where forces might be more evenly matched. This means that as time progresses towards the mid-game, the system prefers to field forces it believes will have a greater advantage over the opponent. In addition, we weight the mineral disparity values such that information gathered more recently has a higher effect on the drive than that gathered less recently. This causes the drive to spike slightly when new information has been gathered. Sustained spikes over a short period of time help to breach the threshold and cause attacks to be triggered.

A. The Synapse system

We were particularly inspired by the work of [4], who provide an algorithm for automated unit training in Real-Time Strategy games based on insect-like scheduling. The aim being to remove this macro responsibility from a player, and allow them to focus limited attentional resources on micro-managing units to better effect. We implement a simpler system with similar goals we call Synapse which uses a drive to periodically instantiate tasks to continually produce units for use by other parts of the system. Through this, our aim is to treat units as a resource that can be requisitioned and utilised by other parts of the system. We also aim to abstract over the need to impose metric constraints on armies – For instance, we would like to avoid entirely the need to specify that an army must contain n units before an attack can be launched. Instead, we view that the motivating forces behind such decisions should be based on qualitative reasoning about relative force strengths and opponent strategies, not entirely based in metric measurements of the number of units available, though this is certainly a factor.

This may be the end of our discussion of this aspect of our system if all we required to be able to do was to produce a large number of a single unit type. In reality, success in RTS games such as Starcraft involves producing and managing heterogeneous teams of units. This then produces more questions - for instance, how do we decide how many units of a certain type to deploy as part of an army?

We look at potential solutions to this problem as a matter of evaluating the qualitative properties of the army. Our particular approach is to describe the composition of an army in terms of a set of relative ratios of unit types. That is to say, we can express that an army should contain, for instance, one Zealot for every three Dragoons. This then allows us to state whether the condition holds or not in terms of the number of those units we possess. If not, we determine which type should be built in order to attempt to bring the ratio back into balance.

We implement a single numeric drive that increases in value over time until a threshold is hit, at which point it is returned to its starting value. The size of the army produced by the system can be controlled by increasing the value of the threshold over time, such that production speed slows over time, resulting in the army reaching some desired, terminal size. However instead we choose to keep the threshold static, meaning that the system will engage in continual, synchronised production across all facilities under its control until the population cap is hit.

Once the drive threshold is hit, the system evaluates the set of unit ratios available to it. Ratios that are unbalanced are filtered, and those with the highest magnitudes are selected. The system then instantiates tasks to bring these ratios back towards their intended values. For the production of military units if a ratio is in perfect balance we instantiate training tasks to produce units to deliberately violate the ratio slightly, so as to motivate continual production. However, in some cases we also wish for a ratio to remain in harmony once it is met (for example, we wish to have three Gateways for every Nexus that we have) which is also possible.

The Synapse system is able to manage the construction of an army across any number of production facilities. The composition of the force produced is defined by the set of unit ratios present in the system. These however are not static and may be modulated by other processes in the system. For instance, the choice of which units to deploy might change due to an opponent fielding units that counter or are particularly effective against those produced so far. We express the transition between distinct army compositions as a modulation of the set of unit ratios, meaning that the composition of an army can be modified in real-time, thus allowing the system to adapt to new situations.

B. Scouting and Learning

We were interested in how a system might learn how to gather information about an opponent. The obvious sources of such information are base locations where the opponent may have buildings and units to observe and evaluate. However, units do not stay in one place as buildings do, and there may exist more than one base location under the control of the opponent. If we consider that units might move around, we might also be interested in where we need to look on the map in order to observe this happening.

We accomplish this by employing an abstraction of the game map based around small chunks of space in the world we call *places*, inspired by the work of [12] on a similar system used for cognitive mapping on mobile robots. A Place can be base locations or choke points, which we extract from the map at the start of a game. An example can be seen in *Figure 3*, where the orange circle represents a place over a large choke point, and *Figure 2* where a place is located over a start location. Each place is initially seeded with a heuristic score, based on its distance from the system's starting location, meaning that those places further away are initially scored more highly. In addition, each place has a value coefficient which acts as a ranking, distinguishing starting locations from other base locations and choke points. In short, through this we define at design-time the parts of the map we believe may initially

be interesting – starting locations, and the choke points and expansion locations around them.

At runtime, when scouts are employed, they select and explore locations from the set of available places, preferring those ranked more highly by the heuristic score. As a scout explores a place, it modifies the place's heuristic score based on how much extra knowledge is gained about the opponent from visiting it. That is to say that observing new units, buildings or tech that might have been as-yet undiscovered, increases the heuristic value associated with a place. However, viewing units that have already been observed and known to exist does not add to the score.

In short, this means that as the system explores the map, it learns which points are interesting to look at in order to learn about the opponent. Those places that yield the most knowledge are preferred targets for re-visiting, in order to keep that knowledge up-to-date. This information is fed into the database of beliefs, and used for various forms of decision-making (such as deciding when to attack) in other parts of the system.

VII. Micro Management

We now describe systems used for micro-level control of individual units.

A. Behaviour-based approach

We implement a micro-management architecture roughly based on the principles of the subsumption architecture of Brooks [13], [14], popularly applied to the control of mobile robots in the last two decades. The system is designed as a layered hierarchy of behaviours, with each behaviour possessing an activation criteria as well as a definition in code. As the activation criteria for a behaviour is met, the behaviour fires and executes the payload of code. This approach is particularly well-suited to the decentralised, reactive control of large numbers of small AI systems. In our system, the behaviour of all units is designed using this general principle, though naturally each unique unit type utilises implementations tailored to its own specific characteristics and needs. For instance, Arbiters have behaviours to seek out crowds for the application of Stasis, whereas Dragoons do not.

Each behaviour in the hierarchy may suppress either those directly below it, or a set. A concrete example of how this is applied is given by an implementation of simple Dark Templar tactics used for harassment. Each Templar has three behaviours - attack, hide and explore, which the system attempts to execute sequentially on each frame. The highest priority behaviour is to hide if detected - at which point a Templar tries to find a safe place where he will not be detected or attacked. That is, this behaviour is evaluated first – if it returns true, its code is executed and all lower-level behaviours are suppressed. Second highest is to attack - if not hiding the Templar will seek out targets in the area. Finally, if no targets are believed to exist, the Templar will attempt to explore the area. As such, the hiding behaviour is able to suppress both the attacking and exploring behaviours, and the attacking behaviour is able to suppress the exploring behaviour.

B. Virtual Emotions

On top of our behaviour-based architecture we also implement a system of virtual drives to provide emergent, emotion-based control of units [15]. Specifically, this is used as a mechanism to equip units with the tools required to make decisions in combat about when to attack or retreat, but primarily to facilitate decentralised combat formations. In this we draw from existing work in Artificial Life systems [16], as well as applications of similar techniques to virtual characters in modern video games [17]. In our system we apply a version of these techniques to all combat units, with each unit type following a set of rules designed to best exploit its unique characteristics.

Managing the formation of units in combat centrally is a difficult task. Ground-based units are constrained by the structure of the environment, meaning that formations must be calculated to take this into account. Maintaining such a formation during movement exasperates the complexity of the task. We preferred then to provide a set of rules that can produce emergent formations in a decentralised way, making the task more manageable, and potentially locating novel structure in the environment that may not be able to be recognised or fully exploited by centralised approaches (for example, by using a set of pre-defined formation templates). Here we discuss the application of such techniques to the Protoss Dragoon.

We assign units two discrete, numeric drives, which we refer to in natural terms as "confidence" and "caution". The drives are modulated by the presence of friendly and enemy buildings and units in the immediate vicinity of the unit at a particular point in time. If a unit is supported by its allies, it will grow confident, but will grow cautious in the presence of enemy units. If a unit is more confident than cautious, it will be more likely to stand its ground and attack nearby enemy units. However, as the drives approach equilibrium the unit will be more likely to begin backing away towards a safer nearby point (for example, a point more well-defended by friendly units or buildings), leading to a full retreat if overwhelmed. These points are located using a particle filter centred on the Dragoon that examines and scores nearby points in terms of their relative caution values. In the case of Protoss Dragoons, this allows us to replicate the "Dragoon Dancing" tactic. We have also applied the same technique to Terran Vultures to replicate Vulture kiting.

We couple these drives with a desire for each unit to ensure that it is never in weapons range of more enemy units than a specified threshold, meaning that units will become significantly more cautious if this condition is violated, though grow more confident if it's own weapons range is greater than that of an opponent. This produces emergent unit formations, such as shown in *Figure 3*. In this example, the force had just completed combat against a group of enemies that were in the centre of the screen. The drive-based system pushed units to

Fig. 3. Emergent arc formation, post-combat.

form themselves in an arc around the opponents, attempting to find equilibrium between the confidence and caution drives, while keeping the number of enemy units in range below the specified threshold, with the result of maximising the area covered by the firepower of the force. Interestingly, these kinds of formations mimic those employed by human players, though an AI system is capable of executing them to higher degrees of precision and scale.

Using this architecture we are able to easily implement more advanced behaviours that support other parts of the system. For instance, units will be less likely to retreat from a battle in which they are defending a central base unit (such as a Protoss Nexus) or a group of workers - as we would prefer that they battle for as long as possible in order to buy time for reinforcements to arrive. We accomplish this behaviour by simply modulating the confidence drive in the presence of such units. The effect is that, while this does not entirely preclude the opportunity for retreat in the face of an overwhelming force, it is however less likely.

C. Path finding: Threat-aware and Space-utilising

We make use of several implementations of threat-aware path-finding, which take advantage of our belief-based architecture in order to find paths through the environment that place a unit in minimal contact with enemy units, such as missile turrets, photon cannons and so on. To do this, we build on the A-Star search implementation found in the source of the BWTA library [18].

We make two important modifications. Firstly, we provide a slider for modulating the granularity of the search, which causes the algorithm to move in larger steps if so desired (ie. only every *n*'th tile being considered). This increases the speed of the algorithm significantly, and means that it can be used efficiently in real-time across short and medium distances with minimal impact on performance. Plans of higher granularity are also useful for directing airborne units, as since they do not have the same constraints on terrain geometry as ground units, do not need highly detailed plans.

Secondly, we modify the heuristic value of each node evaluated during the search based on the number of enemy units it is in range of, information which is extracted from our database of beliefs. This means that such positions are penalised, and paths that navigate around structures such as enemy missile turrets, bunkers, photon cannons etc. are preferred.

This is necessarily performed under the condition of partial observability, however. For instance, we may plan a path into an enemy base, and upon arriving see that the path now crosses into the firing line of a turret, which we had previously not seen. To deal with such situations, we employ an event-based model coupled with a replanning approach. When new emplacements are discovered by the system, all plans currently being followed by units are evaluated to see if any of path points intersect with the firing range of the newly observed unit. If so, we simply replan the path from the current position of the affected unit to the target, avoiding the newly-discovered emplacement. This allows a transport such as a Shuttle to find a route directly into the centre of an enemy base, while safely avoiding anti-air units.

D. Space-Filling

In experiments, we noted significant disparities between the way that human and AI players control groups of units. A human player, when moving an army, will move it gradually, bit-by-bit towards the target area. AI players often issue a single move command towards the target, causing each unit in the army to use the path calculated by its in-game path finding. However, this optimum, shortest-distance path may not be the best choice, as it often results in an army sticking to walls and becoming separated out into a long line of individual units which can be easily picked off.

We view that one difference is as follows: Human players attempt to make maximum use of the space available to them when moving groups of units around. Rigidly sticking to in-game path-finding does not accomplish this.

However, we would prefer to not have to abandon the capabilities of the path-finding algorithm entirely. Our approach then is to calculate a path to our target are as normal, but we then employ a series of particle filters on each step of the path in order to try to improve points to make more use of available space.

This is accomplished as follows. We first calculate the path α to the target using a standard A-Star search. For each position $\rho \in \alpha$ we generate a distribution $f(\rho)$ of particles in the area around it. We then generate a second tier $g(f(\rho))$ of smaller distributions around each of these points. The points generated on this final step are filtered based on whether they lie on a walkable map tile or not, and assigned a score of 1 or 0 respectively. For each point in the original distribution $f(\rho)$, we then assign the sum of the scores of all child points generated by $g(f(\rho))$. Comparing the score of the original path node with that of the highest scoring point generated around it, those path nodes that are capable of being moved to areas where they are surrounded by more walkable points than their original location are swapped with their improved children.

One weakness of the path-finding system is its speed. While the rest of our system is capable of acting in real-time, with a typical "tick" taking around 2-3ms, threat-aware A-Star path-finding often takes upwards of 90ms to locate a path. The primary reason for this is that, when tasked with calculating paths across long distances (such as moving an army from one side of a map to another) our system will calculate a full path from start to finish. In comparison, the built-in path-finding system of Starcraft seems to calculate partial paths to intermediate points, and re-calculates from there once the unit reaches them, making the search more efficient. We should also mention however that our search is further exasperated by having to calculate threat values for each point in the search.

VIII. Evaluation

The most common form of evaluation for Starcraft AIs is to compare work against the default AI players in the game. However the growing sophistication of third party systems means that this task is becoming increasingly trivial. It also places an upper limit on the need for system adaptability, as the in-game AI players have an extremely small space of possible strategies, requiring little learning and adaptation after a point. A more challenging form of evaluation is to play against expert human players, however these are rare and evaluation is necessarily expensive in terms of time, making it difficult to acquire statistically significant results. Somewhere in-between these two extremes lies the possibility of evaluation against existing, third-party AI systems that compete in tournaments such as those hosted by IEEE CIG and AIIDE each year. We argue that this form of evaluation should be standard, as it allows us to further push the boundaries of AI systems in this domain.

IX. Experiment Parameters

We took each of the AIs submitted to the 2011 AIIDE Starcraft AI competition, along with the built-in AI players of Starcraft, and played 500 games with our own system against each, on a random map rotation taken from those used in the AIIDE tournament. We employed a time-limit of 84000 in-game frames, upon which the AI with the highest in-game score would be considered the winner. Crashes were marked as a win for the non-crashing system.

X. Quantitative Analysis

Figures *4* and *5* show our results. Overall we see that while our system performs well against the default AIs of Starcraft, third-party AIs prove to be more of a challenge. This is to be expected, as the sophistication of such systems far outweighs that of the now 14-year old in-game AIs. The set of systems our AI performs more poorly against seems to correlate with those systems that we know to be the state-of-the-art, and that have been subject to extended periods of development, often by teams, and known to perform well in the tournament settings of IEEE CIG and AIIDE (such as Skynet, EISBot and UAlbertaBot).

Name	Win Rate
Zerg	95%
Protoss	92%
Terran	94%

Fig. 4. Comparison of wins between built-in AIs and our system over 300 games.

Name	Race	Win Rate
Aiur	P	32%
bigbrother	Z	82%
BroodwarBotQ	P	42%
BTHAI	Z	62%
Cromulent	T	75%
EISBot	P	37%
ItayUndermind	Z	42%
Nova	T	55%
Quorum	T	90%
Skynet	P	22%
SPAR	P	56%
UAlbertaBot	P	25%
Undermind	T	49%

Fig. 5. Comparison of wins between third party systems and our system over 500 games.

In practice, none of the games played reached the time limit we set, with most games that did not result in a crash lasting between 25,000-35,000 frames.

XI. Qualitative Analysis

We are particularly focused on being able to explain *why* our system performs as it does, so as to develop an ontology of capabilities that might be necessary for good performance. On the few occasions our system did lose to the in-game AIs of Starcraft, we attribute the losses to rare bugs. Considering our system is the result of only a few months of development, we regard this as acceptable. Such bugs include malfunctioning build orders, units becoming stuck and occasional crashes. The particular capability that we observe provides the greatest advantage is that of the emotion-based micro management system. Against systems that do not display as finely-grained control over individual units during combat as our system, such as bigbrother, Cromulent, Quorum and BTHAI, this proves to be highly powerful. AIs that attempt to keep their units bunched up are particularly susceptible to being outgunned by the kind of emergent arc-based structures as in *Figure 3*, as well as systems that move their armies in long, column formations. In addition, our use of a pathfinding algorithm that makes better use of free space than that present in the game allows our armies the room essential for such formations.

A further advantage is provided by our use of particle filters

to control the position of Arbiters by generating a cloud of points and selecting those which allow the Arbiter to stay out of enemy weapons range while still cloaking the units below it, and to manage the trade-off between safety and cloaking. This allows us to quickly and autonomously locate positions in a way that a human player would find extremely difficult while maintaining micro control, especially in cases where more than one Arbiter is present.

Our use of attack drives pays off by ensuring our system is continually aggressive, with even unsuccessful attacks revealing information about the opponent which is later exploited. We observe that several of the systems we evaluated against seem to only attack when a certain, pre-defined number of units is available. Whereas our system attacks when it believes it is able to inflict damage on an opponent, taking into account its own capabilities and a mixture of the observed and predicted strength of an opponent. This produces a scale of possible attacks generated based on this information, from single-unit harassment to large armies. Many systems seem to cope poorly with small, harassing attacks that deal light damage in the early and mid-game. Several systems do not re-build base buildings if they are destroyed. Our own system is able to re-build as needed, as it keeps a logical description of a base's state which is continually monitored, and instantiates construction tasks if violated.

XII. Future Work

Evaluation against human players, as discussed previously, poses several challenges that are outside the scope of our current project, and so this avenue was left for future work, since we were primarily interested in systems capable of competing against other AIs in tournaments. Construction of systems to play against humans requires us to think about different approaches to system design. As AIs are not as constrained by limited attentional resources as humans, but tend to lack learning and adaptation capabilities. In professional tournaments, we observe strategies, both micro and macro, falling in and out of fashion, with new innovations being made each year. The cognitive and information-processing requirements to produce these capabilities are not trivial, and provide a rich source of inspiration for future work.

As far as our own work goes, we are currently making use of our clustering algorithm as a basis for applying Regional Connection Calculi [19] to provide logics for reasoning about the structure and motion of groups of units in qualitative ways. Recent work by Sridhar and Cohn [20] has shown that this approach can provide promising avenues for addressing the task recognition problem, and we are using similar techniques with the aim of producing expressive and re-usable methods of opponent modelling.

XIII. Conclusion

We presented our SCAIL system, an integrated system capable of playing a full round of the Real-Time Strategy game Starcraft. The system makes use of particle filters, online machine learning, drive-based motivation systems and artificial emotions to provide control systems for high and low-level behaviour. We showed that the use of these techniques can provide an advantage over many current state-of-the-art systems. Our system does however still struggle against those AIs that have typically ranked at or near the top of recent tournaments. However, after only a few months of development, we view our results as encouraging, being only at the beginning of our exploration of how these techniques might be best exploited in this interesting domain.

References

[1] G. Synnaeve, "A Bayesian model for RTS units control applied to StarCraft," *Computational Intelligence and Games*, pp. 190–196, 2011.

[2] A. Shantia, E. Begue, and M. Wiering, "Connectionist reinforcement learning for intelligent unit micro management in StarCraft," *International Joint Conference on Neural Networks*, 2011.

[3] I. Gonzalez and L. Garrido, "Spatial Distribution through Swarm Behavior on a Military Group in the Starcraft Video Game," *10th Mexican International Conference on Artificial Intelligence*, pp. 77–82, Nov. 2011.

[4] M. Santos and C. Martinho, "Wasp-Like Scheduling for Unit Training in Real-Time Strategy Games," *AIIDE*, pp. 195–200, 2008.

[5] J. Lewis, "A Corpus Analysis of Strategy Video Game Play in Starcraft: Brood War," *Annual Conference of the Cognitive Science Society*, pp. 687–692, 2011.

[6] B. Weber, "Integrating Expert Knowledge and Experience," *Fifteenth AAAI/SIGART Doctoral Consortium*, 2010.

[7] B. Weber and M. Mateas, "Applying Goal-Driven Autonomy to StarCraft," *Artificial Intelligence and Interactive Digital Entertainment*, no. Orkin, 2010.

[8] A. S. Arnaud Doucet, Nando de Freitas, Neil Gordon, *Sequential Monte Carlo Methods in Practice*, 2001.

[9] M. Ester, H. Kriegel, and J. Sander, "A density-based algorithm for discovering clusters in large spatial databases with noise," *Proc. of 2nd International Conference on Knowledge Discovery and Data Mining*, 1996.

[10] L. P. Beaudoin and A. Sloman, "A Study of Motive Processing and Attention," pp. 229–238, 1993.

[11] M. Hanheide and N. Hawes, "A framework for goal generation and management," *AAAI Workshop on Goal-Directed Autonomy*, 2010.

[12] A. Pronobis, K. Sjoo, and A. Aydemir, "A framework for robust cognitive spatial mapping," *Proceedings of the 14th International Conference on Advanced Robotics*, 2009.

[13] R. Brooks, "A robust layered control system for a mobile robot," *IEEE Journal of Robotics and Automation*, 1986.

[14] ——, "Intelligence without representation," *Artificial intelligence*, 1991.

[15] M. Scheutz, "Useful roles of emotions in artificial agents: A case study from artificial life," *Proceedings of the 19th national conference on Artifical intelligence*, pp. 42—-47, 2004.

[16] M. Ptaszynski, "A Pragmatic Approach to Implementation of Emotional Intelligence in Machines," pp. 101–102, 2008.

[17] L. Pena, S. Ossowski, J. Pena, and J. Sanchez, "EEP A lightweight emotional model : Application to RPG video game characters," in *IEEE Computational Intelligence and Games*. IEEE, 2011, pp. 142–149.

[18] L. Perkins, "Terrain Analysis in Real-Time Strategy Games: An Integrated Approach to Choke Point Detection and Region Decomposition," *AIIDE*, pp. 168–173, 2010.

[19] D. Randell, Z. Cui, and A. Cohn, "A spatial logic based on regions and connection," *KR*, 1992.

[20] M. Sridhar and A. Cohn, "Unsupervised learning of event classes from video," *AAAI*, pp. 1631–1638, 2010.