A Bayesian Model for RTS Units Control applied to StarCraft

Gabriel Synnaeve (gabriel.synnaeve@gmail.com) Pierre Bessière (pierre.bessiere@imag.fr)

Abstract—In real-time strategy games (RTS), the player must reason about high-level strategy and planning while having effective tactics and even individual units micro-management. Enabling an artificial agent to deal with such a task entails breaking down the complexity of this environment. For that, we propose to control units locally in the Bayesian sensory motor robot fashion, with higher level orders integrated as perceptions. As complete inference encompassing global strategy down to individual unit needs is intractable, we embrace incompleteness through a hierarchical model able to deal with uncertainty. We developed and applied our approach on a StarCraft[1] AI.

I. Introduction

In video games, AI is an increasingly central part of the gameplay: foes in every single player game of course, but also assisting squads in first person shooters (FPS), neutral and assisting non-playing characters (NPC) in role playing games (RPG and MMORPG), and autonomy of units controlled by the player in real-time strategy (RTS), so that he can focus more on strategy and less on control. Multi-player is not going to end video games need for intelligent behavior of NPC. Even for gameplay oriented or training purposes, "bots" have a great future and room to improve.

RTS gameplay consist in gathering resources, building up an economic and military power through growth and technology, to defeat your opponent by destroying his base, army and economy. It requires dealing with strategy, tactics, and units management (often called micro-management) in real-time. Strategy consist in what will be done in the long term as well as predicting what the enemy is doing. It particularly deals with the economy/army trade-off estimation, army composition, long-term planning. The three aggregate indicators for strategy are aggression, production, and technology. The tactical aspect of the gameplay is dominated by military moves: when, where (with regard to topography and weak points), how to attack or defend. This implies dealing with *extensional* (what the invisible units under "fog of war" are doing) and *intentional* (what will the visible enemy units do) uncertainty.

In this paper, we focus on micro-management, which is the art of maximizing the effectiveness of the units *i.e.* the damages given/damages received ratio. For instance: retreat and save a wounded unit so that the enemy units would have to chase it either boosts your firepower or weakens the opponent's. In the field of units control, the dimension of the set of possible actions each micro-turn (for instance: 1/24th of a second in StarCraft) constrains reasoning about the state of the game to be hierarchical, with different levels of granularity. In most RTS games, a unit can go (at least) in its

24 surrounding tiles (see Figure 3, combination of N, S, E, W up to the 2nd order), stay where it is, attack, and sometimes cast different spells: more than 26 possible actions each turn. Even if we consider only 8 possible directions, stay, and attack, with N units, there are 10^N possible combinations each turn (all units make a move each turn). As large battles in StarCraft account for *at least* 20 units on each side, optimal units control hides in too big a search space to be fully explored in real-time (sub-second reaction at least) on normal hardware, even if we take only one decision per unit per second.

We present a distributed sensory-motor model for micro-management, able to handle both the complexity of unit control and the need of hierarchy (see Figure 1). This paper focuses on the part inside the dotted line. We treat the units independently, thus reducing the complexity (no communication between "Bayesian units"), and allows to take higher-level orders into account along with local situation handling. For instance: the tactical planner may decide to retreat, or go through a choke under enemy fire, each Bayesian unit will have the higher-level order as a sensory input, along with topography, foes and allies positions. From its perception, our Bayesian robot [1] can compute the distribution over its motor control. The sensory inputs given to a "Bayesian unit" controls its objective(s) or goal(s) and the parametrization of his probabilistic model controls its behavior and degree of freedom. As an illustration (only), two of the extreme cases are $P(Direction = x|Objective = x) = 1$: no freedom, $P(Direction = x|Objective = y) = P(Direction = x)$: no influence of the objective. The performances of our models are evaluated against the original StarCraft AI and a reference AI and have proved excellent in this benchmark setup.

II. Related Work

Video games AI research is yielding new approaches to a wide range of problems, for instance in RTS: pathfinding, multiple agents coordination, collaboration, prediction, planning and (multi-scale) reasoning under uncertainty. These problems are particularly interesting in the RTS framework because the solutions have to deal with many objects, imperfect information and micro-actions while running in real-time on desktop hardware. Technical solutions include finite states machines (FSM) [2], genetic algorithms (GA) [3], [4], reinforcement learning (RL) [5], [6], case-based reasoning (CBR) [7], [8], continuous action models [9], reactive planning [10], upper confidence bounds tree (UCT) [11], potential fields [12], influence maps[4], and cognitive human-inspired models [13].

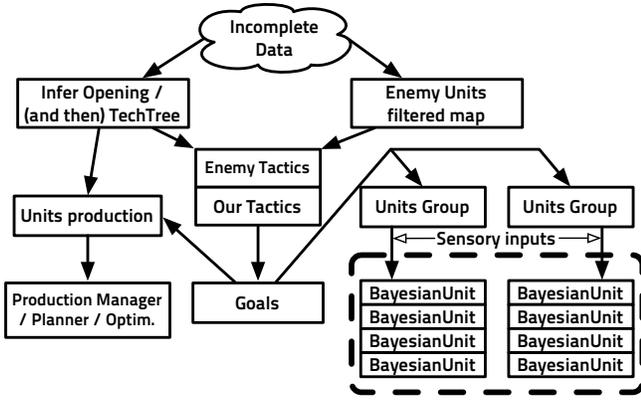[1]StarCraft and its expansion StarCraft: Brood War are trademarks of Blizzard Entertainment[TM]

Fig. 1. Data centric overview of the StarCraft bot player, the part presented in this paper is the one of the "BayesianUnit" (inside dotted lines).

FSM are well-known and widely used for control tasks due to their efficiency and implementation simplicity. However, they don't allow for state sharing, which increases the number of transitions to manage, and state storing, which makes collaborative behavior hard to code [14]. Hierarchical FSM (HFSM) solve some of this problems (state sharing) and evolved into behavior trees (BT, hybrids HFSM) [15] and behavior multi-queues (resumable, better for animation) [14] that conserved high performances. However, adaptivity of behavior by parameters learning is not the main focus of these models, and unit control is a task that would require a huge amount of hand tuning of the behaviors to be really efficient. Also, these architectures does not allow reasoning under uncertainty, which helps dealing with local enemy and even allied units. Our agents see local enemy (and allied) units but do not know what action they are going to do. They could have perfect information about the allied units intentions, but this would need extensive communication between all the units.

Some interesting uses of RL [16] to RTS research are concurrent hierarchical (units Q-functions are combined higher up) RL [5] to efficiently control units in a multi-effector system fashion, and large-scale strategy games [6]. In real game setups, RL models have to deal with the fact that the state spaces to explore is enormous, so learning will be slow or shallow. It also requires the structure of the game to be described in a partial program (or often a partial Markov decision process) and a shape function [5]. RL is a transversal technique to learn parameters of an underlying model, and this underlying behavioral model matters. The same problems arise with evolutionary learning techniques [3].

Case-based reasoning (CBR) allows for learning against dynamic opponents [7] and has been applied successfully to strategic and tactical planning down to execution through behavior reasoning rules [17]. CBR limitations (as well as RL) include the necessary approximation of the world and the difficulty to work with multi-scale goals and plans. These

problems led respectively to continuous action models [9], an integrated RL/CBR algorithm using continuous models, and reactive planning [10], a decompositional planning similar to hierarchical task networks in that sub-plans can be changed at different granularity levels. Reactive planning allows for multi-scale (hierarchical) goals/actions integration and has been reported working on StarCraft, the main drawback is that it does not address uncertainty and so can not simply deal with hidden information (both extensional and intentional). Fully integrated FSM, BT, RL and CBR models all need vertical integration of goals, which is not very flexible (except in reactive planning).

Monte-Carlo planning [18] and upper Upper confidence bounds tree (UCT) planning (coming from Go AI) [11] samples through the (rigorously intractable) plans space by incrementally building the actions tree through Monte-Carlo sampling. UCT for tactical assault planning [11] in RTS does not require to encode human knowledge (by opposition to Monte-Carlo planning) but it is very costly, both in learning and running time, to go down to units control on RTS problems. Our model subsumes potential fields [12], which are powerful and used in new generation RTS AI to handle threat, as some of our Bayesian unit sensory inputs are potential damages and tactical goodness (height for the moment) of positions. Our model provides flocking and local (subjective to the unit) influences on the pathfinding as in [4]. In their paper, Preuss *et al.* are driven by the same quest for a more natural and efficient behavior for units in RTS. Finally, there are some cognitive approaches to RTS AI [13], and we particularly agree with Wintermute *et al.* analysis of RTS AI problems. Our model has some similarities: separate and different agents for different levels of abstraction/reasoning and also a perception-action approach (see Figure 1).

III. BAYESIAN PROGRAMMING

We introduce Bayesian programs (BP), a formalism that can be used to describe entirely any kind of Bayesian model, subsuming Bayesian networks and Bayesian maps, equivalent to probabilistic factor graphs [19]. There are mainly two parts in a BP, the **description** of how to compute the joint distribution, and the **question(s)** that it will be asked.

The description consists in explaining the relevant *variables* $\{X^1, \ldots, X^n\}$ and explain their dependencies by *decomposing* the joint distribution $P(X^1 \ldots X^n | \delta, \pi)$ with existing preliminary knowledge π and data δ. The *forms* of each term of the product specify how to compute their distributions: either parametric forms (laws or probability tables, with free parameters that can be learned from data δ) or recursive questions to other Bayesian programs.

Answering a question is computing the distribution $P(Searched|Known)$, with $Searched$ and $Known$ two disjoint subsets of the variables. $P(Searched|Known)$

$$= \frac{\sum_{Free} P(Searched, \ Free, \ Known)}{P(Known)}$$

$$= \frac{1}{Z} \times \sum_{Free} P(Searched, \ Free, \ Known)$$

General Bayesian inference is practically intractable, but conditional independence hypotheses and constraints (stated in the description) often simplify the model. Also, there are different well-known approximation techniques, for instance Monte Carlo methods [20] and variational Bayes [21]. In this paper, we will use a specific fusion model (inverse programming) that allows for complete inference to be computed in real-time.

$$
BP \begin{cases} Desc. \begin{cases} Spec.(\pi) \begin{cases} Variables \\ Decomposition \\ Forms \ (Parametric \ or \ Program) \end{cases} \\ Identification \ (based \ on \ \delta) \end{cases} \\ Question \end{cases}
$$

For the use of Bayesian programming in sensory-motor systems, see [22]. For its use in cognitive modeling, see [23]. For its first use in video games (first person shooter gameplay, Unreal Tournament), see [24].

IV. A MODEL FOR UNIT CONTROL

A. A Simple Top-Down Solution

How do we set the reward or value function for micro-management: is staying alive better than killing an enemy unit? Even if we could compute to the end of the fight and/or apply the same approach that we have for board games, how do we infer the best "set of next moves" for the enemy when the space of possible moves is so huge and the number of possible reasoning methods (sacrifices and influences of other parts of the game for instance) is bigger than for Chess? As complete search through the min/max tree, if there exists such thing in a RTS, is intractable, we propose a greedy target selection heuristic leading the movements of units to benchmark our Bayesian model against. In this solution, each unit can be viewed as an effector, part of a multi-body (multi-effector) agent. Let $\mathbf{U}$ be the set of the m units to control, $\mathbf{A} = \mathbf{D} \cup \mathbf{S}$ be the set of possible actions (all n possible **D**irections, standing ground included, and **S**kills, firing included), and $\mathbf{E}$ the set of enemies. As $|\mathbf{U}| = m$, we have $|\mathbf{A}|^m$ possible combinations each turn, and the enemy has $|\mathbf{A}|^{|\mathbf{E}|}$.

The idea behind the heuristic used for target selection is that units need to focus fire (less incoming damages if enemy units die faster) on units that do the most damages, have the less hit points, and take the most damages from their attack type. This can be achieved by using a data structure, shared by all our units engaged in the battle, that stores the damages corresponding to future allied attacks for each enemy units. Whenever a unit will fire on a enemy unit, it registers there the future damages on the enemy unit. We also need a set of priority targets for each of our unit types that can be drawn from expert knowledge or learned from reinforcement learning battling all unit types. A unit select its target among the most focus fired units with positive future hit point (current hit points minus registered damages), while prioritizing units from the priority set of its type. The units

group can also impose its own priorities on enemy units (for instance to achieve a goal).

The only degenerated case would be if all our units register their targets at once (and all the enemy units have the same priority) and it never happens (plus, units fire rates have a randomness factor). Indeed, our Bayesian model uses this target selection heuristic, but that is all both models have in common. From there, units are controlled with a very simple FSM: fire when possible (weapon reloaded and target in range), move towards target when out of range.

B. Our Model: a Bayesian Bottom-Up Solution

We use Bayesian programming as an alternative to logic, transforming incompleteness of knowledge about the world into uncertainty. In the case of units management, we have mainly *intensional* uncertainty. Instead of asking questions like: where are other units going to be next frame? 10 frames later? Our model is based on rough estimations that are not taken as ground facts. Knowing the answer to these questions would require for our own (allied) units to communicate a lot and to stick to their plan (which does not allow for quick reaction nor adaptation). For enemy units, it would require exploring the tree of possible plans (intractable) whose we can only draw samples from [11]. Even so, taking enemy minimax (to which depth?) moves for facts would assume that the enemy is also playing minimax (to the same depth) following exactly the same valuation rules as ours. Clearly, RTS micro-management is more inclined to reactive planning than board games reasoning. That does not exclude having higher level (strategic and tactic) goals. In our model, they are fed to the unit as sensory inputs, that will have an influence on its behavior depending on the situation/state the unit is in.
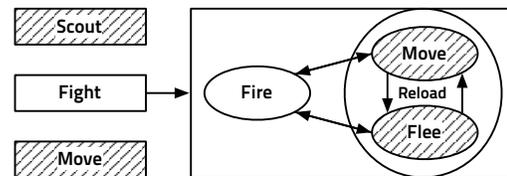

Fig. 2. Bayesian unit modal FSM, detail on the fight mode. Stripped modes are Bayesian.

We propose to model units as sensory-motor robots described within the Bayesian robot programming framework [1]. A Bayesian model uses and reasons on distributions instead of predicates, which deals directly with uncertainty. Our Bayesian units are simple hierarchical finite states machines (states can be seen as modes) that can scout, fight and move (see Figure 2). Each unit type has a reload rate and attack duration, so their fight mode will be like:

if $canFire \wedge t = selectTarget() \wedge inRange(t)$ **then**
  $attack(t)$
else if $needFlee()$ **then**

```
    flee()
else
    fightMove()
end if
```

The unit needs to determine where to go when fleeing and moving during a fight, optimally with regard to its target and the attacking enemies, while avoiding collisions (which results in blocked units and time lost) as much as possible. $flee()$ and $fightMove()$ call the Bayesian model (expressed in Bayesian programming, see section 3.) that follows:

1) Variables:

- $Dir_{i \in [\![0...n]\!]} \in \{True, False\}$: at least one variable for each atomic direction the unit can go to. $P(Dir_i = True) = 1$ (also noted $P(Dir_i) = 1$) means that the unit will certainly go in direction i ($\Leftrightarrow \mathbf{D}[i]$). For example, in StarCraft we use the 24 atomic directions (48 for the smallest and fast units as we use a proportional scale) plus the current unit position (stay where it is) as shown in Figure 3. We could use one variable with 24 directions, the approach would be the same.

- $Obj_{i \in [\![0...n]\!]} \in \{True, False\}$: direction of the objective (given by a higher rank model). $P(Obj_i) = 1$ means that the direction i is totally in the direction of the objective (move, retreat or offensive position computed by the strategic or tactical manager). In our StarCraft AI, we use the scalar product between the direction i and the objective vector (output of the pathfinding) with a minimum value of 0.01 so that the probability to go in a given direction is proportional to its alignment with the objective. Note that some situation have a null objective (the unit is free to move).

- $Dmg_{i \in [\![0...n]\!]} \in [DamageValues]$ for instance, with $ubhp$ standing as unit base hit points, $Dmg_i \in \left\{0, [\![0 \ldots \frac{ubhp}{2}]\!],]\!]\frac{ubhp}{2} \ldots ubhp[\![, [\![ubhp \cdots + \inf[\![\right\}$. This will act as subjective potential fields [12] in which the (repulsive) influence of the potential damages map depends on the unit type. In our StarCraft AI, this is directly drawn from two constantly updated potential damage maps (air, ground). For instance, it allows our scouting units to avoid potential attacks as much as possible.

- $A_{i \in [\![0...n]\!]} \in \{None, Small, Big\}$: occupation of the direction i by a allied unit. The model can effectively use many values (other than "occupied/free") because directions may be multi-scale (for instance we indexed the scale on the size of the unit) and, in the end, small and/or fast units have a much smaller footprint, collision wise, than big and/or slow. In our AI, instead of direct positions of allied units, we used their (linear) interpolation $\frac{|unit,\mathbf{D}[i]|}{unit\_speed}$ frames later (to avoid squeezing/expansion).

- $E_{i \in [\![0...n]\!]} \in \{None, Small, Big\}$: occupation of direction i by a enemy unit. As above.

- $Occ_{i \in [\![0...n]\!]} \in \{None, Building, StaticTerrain\}$ (this could have been 2 variables or we could omit static terrain but we stay as general as possible): repulsive

effect of buildings and terrain (cliffs, water, walls). There is basically one set of (sensory) variables per effect in addition to the Dir_i values. In general, if one decides to cover a lot of space with directions (*i.e.* have more than just atomic directions, *i.e.* use this model for planning), one needs to consider directions whose paths collide with each others. For instance, a $\mathbf{D}[i]$ far from the unit can force the unit to go through a wall of allied units ($A_j = Big$) or potential damages.

2) Decomposition: The joint distribution (JD) over these variables is a specific kind of fusion called inverse programming [24]. The sensory variables are considered independent knowing the actions, contrary to standard naive Bayesian fusion, in which the sensory variables are considered independent knowing the phenomenon.

$$P(Dir_{1:n}, Obj_{1:n}, Dmg_{1:n}, A_{1:n}, E_{1:n}, Occ_{1:n})$$

$$= JD = \prod_{i=1}^{n} \quad P(Dir_i)$$
$$P(Obj_i | Dir_i)$$
$$P(Dmg_i | Dir_i)$$
$$P(A_i | Dir_i)$$
$$P(E_i | Dir_i)$$
$$P(Occ_i | Dir_i)$$

We assume that the i directions are independent depending on the action because dependency is already encoded in (all) sensory inputs. We do not have $P(Obj_i) = 1, P(Obj_{j \neq i}) = 0$ but a "continuous" function on i for instance.

3) Forms:

- $P(Dir_i)$ prior on directions, unknown, so unspecified/uniform over all i. $P(Dir_i) = 0.5$.

- $P(Obj_i | Dir_i)$ for instance, "probability that this direction is the objective knowing that we go there" $P(Obj_i = T | Dir_i = T)$ is very high (close to one) when rushing towards an objective, whereas it is far less important when fleeing. Probability table: $P(Obj_i | Dir_i) = table[obj, dir]$

- $P(Dmg_i | Dir_i)$ probability of damages values in some direction knowing this is the unit direction. $P(Dmg_i \in [ubhp, + \inf[| Dir_i = T)$ has to be small in many cases. Probability table.

- $P(A_i | Dir_i)$ probability table that there is an ally in some direction knowing this is the unit direction. Used to avoid collisions.

- $P(E_i | Dir_i)$ probability table, same as above with enemy units, different parameters as we may want to be stucking enemy units, or avoid them.

- $P(Occ_i | Dir_i)$ probability table that there is a blocking building or terrain element is some direction, knowing this is the unit direction, $P(Occ_i = Static | Dir_i = T)$ will be very low (0), whereas $P(Occ_i = Building | Dir_i = T)$ will also be very low but triggers building attack (and destruction) when there are no other issues.

4) Additional variables: There are additional variables for specific modes/behaviors:

- $Prio_{i \in [\![0...n]\!]} \in \{True, False\}$: combined effect of the priority targets that attract the unit while in fight ($fightMove()$). The JD is modified as $JD \times \Pi_{i=1}^n P(Prio_i | Dir_i)$, where $P(Prio_i | Dir_i)$ is a probability table, that corresponds to the attraction of a priority (maybe out of range) target in this direction. This is efficient to be able to target casters or long range units for instance.

- $Att_{i \in [\![0...n]\!], j \in [\![0...m]\!]}$: allied units attractions and repulsions to produce a *flocking* behavior while moving. Different than A_i, the JD would become $JD \times \Pi_{i=1}^n \Pi_{j=1}^m P(Att_{i,j} | Dir_i)$, where $P(Att_{i,j} | Dir_i)$ is a probability table for flocking: a too close unit j will repel the Bayesian unit ($P(Att_{i,j} | Dir_i) < mean$) whereas another unit j will attract depending on its distance (and possibly, leadership).

- $Dir_{i \in [\![0...n]\!]}^{t-1} \in \{True, False\}$: the previous selected direction, $Dir_i^{t-1} = T$ iff the unit went to the direction i, else $False$ for a steering (smooth) behavior. The JD would then be $JD \times \Pi_{i=1}^n P(Dir_i^{t-1} | Dir_i)$, with $P(Dir_i^{t-1} | Dir_i)$ the influence of the last direction, either a table or a parametrized Bell shape over all the i.

- One can have a distribution over a n valued variable $Dir \in \{D\}$. (Each set of boolean random variable can be seen as a $|D|$ valued variable.) The JD would then be $JD \times P(Dir).\Pi_{i=1}^n P(Dir_i | Dir)$.

5) Identification: Parameters and probability tables can be learned through reinforcement learning [16], [25] by setting up different and pertinent scenarii and search for the set of parameters that maximizes a reward function. In our current implementation, the parameters and probability table values are mainly hand specified.

6) Question: When in $fightMove()$, the unit asks:

$$P(Dir_{1:n} | Obj_{1:n}, Dmg_{1:n}, A_{1:n}, E_{1:n}, Occ_{1:n}, Prio_{1:n})$$

When in $flee()$ or while moving or scouting (different balance/parameters), the unit asks:

$$P(Dir_{1:n} | Obj_{1:n}, Dmg_{1:n}, A_{1:n}, E_{1:n}, Occ_{1:n})$$

When flocking, the unit asks:

$$P(Dir_{1:n} | Obj_{1:n}, Dmg_{1:n}, A_{1:n}, E_{1:n}, Occ_{1:n}, Att_{1:n,1:m})$$

From there, the unit can either go in the most probable Dir_i or sample through them. We describe the effect of this choice in the next section.

V. RESULTS: BROODWARBOTQ

A. StarCraft

StarCraft is a canonical RTS game, as Chess is to board games, because it had been around since 1998, it sold 10 millions licenses and was the best competitive RTS. There are numerous international competitions (World Cyber Games, Electronic Sports World Cup, BlizzCon, IeSF Invitational,
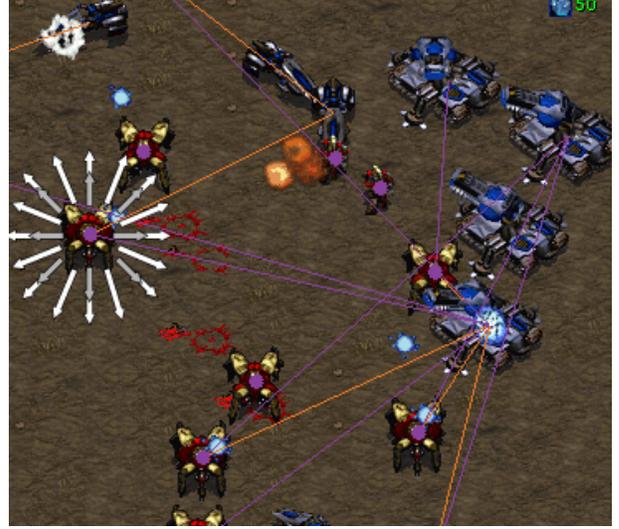

Fig. 3. Screen capture of a fight in which our bot controls the bottom-left units in StarCraft. The 24 possible directions are represented for a unit with white and grey arrows.

OSL, MSL). In *South Korea*, 4.5 millions of licenses have been sold and the average salary of a pro-gamer was up to 4 times the average salary. StarCraft helped define a particular genre of RTS gameplay, based as much on the strategy than the tactics. As a result of that, StarCraft human players are among the best human players, there are a lot of replays available (enabling data-mining and machine learning), and there are tournaments between AIs.

StarCraft micro-management involves ground, flying, ranged, contact, static, moving (at different speeds), small and big units (see Figure 3). Units may also have splash damage, spells, and different types of damages whose amount will depend on the target size. It yields a rich states space and needs control to be very fast: human progamers can perform up to 400 "actions per minute" in intense fights. The problem for them is to know which actions are effective and the most rewarding to spend their actions efficiently. A robot does not have such physical limitations, but yet, badly chosen actions have negative influence on the issue of fights.

B. Our Robot Architecture

Our full robot has separate agents types for separate tasks (strategy, tactics, economy, army, as well as enemy estimations and predictions): the part that interests us here, the unit control, is managed by Bayesian units directly. They take their orders from tactical goals *indirectly* by units groups (see Figure 1), which ares the military goalwise atomic entity regrouping several Bayesian units, tuning their modes (scout, fight, move) and giving them Obj_i as sensory inputs. The Bayesian unit is the smallest entity and controls individual units as sensory-motor robots according to the model described above. The only inter Bayesian units communication about attack targets is handled by a structure shared at the units group level.

C. Experiments

Our implementation[2] (BSD licensed) uses BWAPI[3] to get information from and to control StarCraft. We produced three different AI to run experiments with, along with the original AI (OAI) from StarCraft:

- Heuristic only AI (HOAI), section 4.1: this AI shares the target selection heuristic with our other AI and will be used as a dummy reference (in addition to StarCraft original AI) to avoid bias due to the target selection heuristic.
- Bayesian AI picking best (BAIPB): this AI follows the model of section 4.2 and selects the most probable Dir_i as movement.
- Bayesian AI sampling (BAIS): this AI follows the model of section 4.2 and samples through Dir_i according to their probability ($\Leftrightarrow$ according to Dir distribution).

The experiments consisted in having the AIs fight against each others on a micro-management scenario with mirror matches of 12 and 36 ranged ground units (Dragoons). In the 12 units setup, the unit movements during the battle is easier (less collision probability) than in the 36 units setup. We instantiate only the army manager (no economy in this special maps), one units group manager and as many Bayesian units as there are units provided to us in the scenario. The results are presented in Figure 3.

12 units \ 36 units	OAI	HOAI	BAIPB	BAIS
OAI	(50%)	64%	9%	3%
HOAI	59%	(50%)	11%	6%
BAIPB	93%	97%	(50%)	3%
BAIS	93%	95%	76%	(50%)

Fig. 4. Win ratios over at least 200 battles of OAI, HOAI, BAIPB and BAIS in two mirror setups: 12 and 36 ranged units. Read line vs column: for instance HOAI won 59% of its matches against OAI in the 12 units setup. Note: The average amount of units left at the end of battles is grossly proportional to the percentage of wins.

These results show that our heuristic (HAOI) is comparable to the original AI (OAI), perhaps a little better, but induces more collisions. For Bayesian units however, the "pick best" (BAIPB) direction policy is very effective when battling with few units (and few movements because of static enemy units) as proved against OAI and HOAI, but its effectiveness decreases when the number of units increases: all units are competing for the best directions (to $flee()$ or $fightMove()$ in) and they collide. The sampling policy (BAIS) has way better results in large armies, and significantly better results in the 12 units vs BAIPB, supposedly because BAIPB moves a lot (to chase wounded units) and collide with BAIS units. Sampling entails that the competition for the best directions is distributed among all the "bests to good" wells of well-being, from the units point of view. We also ran tests in

setups with flying units in which BAIPB fared as good as BAIS (no collision for flying units) and way better than OAI.

D. Uses and extensions

This model is currently at the core of the micro-management of our StarCraft bot. We use it mainly with four modes corresponding to four behaviors (four sets of parameters):

- Scout: in this mode, the (often quick and low hit points) unit avoids danger by modifying locally its pathfinding-based, objectives oriented route to avoid damages according to $P(Dmg_i|Dir_i)$.
- In position: in this mode, the unit try to keep its ground but can be "pushed" by other units wanting to pass through with $P(A_i|Dir_i)$. This is useful at a tactical level to do a wall of units that our units can traverse but the opponent's cannot. Basically, there is an attraction to the position of the unit and a stronger repulsion of the interpolation of movements of allied units.
- Flock: in this mode, our unit moves influenced by other allied units through $P(Att_{i\in[\![0...n]\!],j\in[\![0...m]\!]})$ that repulse or attract it depending on its distance to the interpolation of the allied unit j. It allows our units to move more efficiently by not splitting around obstacles and colliding less.
- Fight: in this mode, our unit will follow the damages gradient to smart positions, for instance close to tanks (they cannot fire too close to their position) or far from too much contact units if our unit can attack with range. Our unit moves are also influenced by its priority targets, its goal (go through a choke, flee, etc.) and other units.

This model can be used to specify the behavior of units in RTS games. Instead of relying on a "units push each other" physics model for handling dynamic collision of units, this model makes the units react themselves to collision in a more realistic fashion (a marine cannot push a tank, the tank will move). The realism of units movements can also be augmented with a simple-to-set $P(Dir^{t-1}|Dir^t)$ steering parameter, although we do not use it in the competitive setup.

If we learn the parameters of such a model to mimic existing data (data mining) or to maximize a reward function (reinforcement learning), we can interpret the parameters that will be obtained more easily than parameters of an artificial neural network for instance. Parameters learned in one setup can be reused in another if they are understood.

Finally, we claim that specifying or changing the behavior of this model is much easier than changing the behavior generated by a FSM, and game developers can have a fine control over it. Dynamic switches of behavior (as we do between the scout/flock/inposition/fight modes) are just one probability tables switch away. In fact, probability tables for each sensory input (or group of sensory inputs) can be linked to sliders in a "behavior editor" and game makers can specify the behavior of their units by specifying the degree of effect of each perception (sensory input) on the behavior of the unit

and see the effect in real time. This is not restricted to RTS and could be applied to RPG and even FPS gameplays.

VI. CONCLUSION

In this paper, we have presented a Bayesian model for controlling RTS units locally. We have implemented this model in StarCraft, and it outperforms the original AI as well as other bots (we had a tie with the winner of AIIDE 2010 StarCraft competition, winning with ranged units and losing with contact units). Our approach does not require vertical integration of higher level goals, as opposed to CBR and planning [17], [10], it can have a completely different model above feeding sensory inputs like Obj_i. It scales well with the number of units to control thanks to the absence of communication at the unit level, and is more robust and maintainable than a FSM [2].

Future work will consist in using reinforcement learning to learn the probability tables [16]. It should enhance the performance of our Bayesian units in specific setups. It implies making up challenging scenarii and dealing with huge sampling spaces [25]. We could use multi-modality [23] and inverse programming [24] to get rid of the remaining (small: fire-retreat-move) FSM. Finally, there are yet many collision cases that remain unsolved (particularly visible with contact units like Zealots and Zerglings), so we could also try:

- adding local priority rules to solve collisions (for instance through an asymmetrical $P(Dir_i^{t-1}|Dir_i)$) that would entails units crossing lines with a preferred side (some kind of "social rule"),
- use a units group level supervision using Bayesian units' distributions over Dir as preferences or constraints (for a solver),
- use $P(Dir)$ as an input to another Bayesian model at the units group level of reasoning.

REFERENCES

[1] O. Lebeltel, P. Bessière, J. Diard, and E. Mazer, "Bayesian robot programming," *Autonomous Robots*, vol. 16, no. 1, pp. 49–79, 2004.

[2] S. Rabin, "Implementing a state machine language," *AI Game Programming Wisdom*, pp. 314–320, 2002.

[3] M. Ponsen and I. P. H. M. Spronck, "Improving adaptive game AI with evolutionary learning," in *University of Wolverhampton*, 2004, pp. 389–396.

[4] M. Preuss, N. Beume, H. Danielsiek, T. Hein, B. Naujoks, N. Piatkowski, R. Ster, A. Thom, and S. Wessing, "Towards intelligent team composition and maneuvering in real-time strategy games," *Transactions on Computational Intelligence and AI in Games (CIG)*, vol. 2, no. 2, pp. 82–98, June 2010.

[5] B. Marthi, S. Russell, D. Latham, and C. Guestrin, "Concurrent hierarchical reinforcement learning," in *International Joint Conference of Artificial Intelligence, IJCAI*, 2005, pp. 779–785.

[6] C. Madeira, V. Corruble, and G. Ramalho, "Designing a reinforcement learning-based adaptive AI for large-scale strategy games," in *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*, 2006.

[7] D. W. Aha, M. Molineaux, and M. J. V. Ponsen, "Learning to win: Case-based plan selection in a real-time strategy game," in *ICCBR '05*, 2005, pp. 5–20.

[8] M. Sharma, M. Holmes, J. Santamaria, A. Irani, C. L. Isbell, and A. Ram, "Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL," in *International Joint Conference of Artificial Intelligence, IJCAI*, 2007.

[9] M. Molineaux, D. W. Aha, and P. Moore, "Learning continuous action models in a real-time strategy strategy environment," in *FLAIRS Conference*, 2008, pp. 257–262.

[10] B. G. Weber, P. Mawhorter, M. Mateas, and A. Jhala, "Reactive planning idioms for multi-scale game AI," in *IEEE Symposium on Computational Intelligence and Games (CIG)*, 2010.

[11] R.-K. Balla and A. Fern, "Uct for tactical assault planning in real-time strategy games," in *International Joint Conference of Artificial Intelligence, IJCAI*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2009, pp. 40–45.

[12] J. Hagelbäck and S. J. Johansson, "A multiagent potential field-based bot for real-time strategy games," *Int. J. Comput. Games Technol.*, vol. 2009, pp. 4:1–4:10, January 2009.

[13] S. Wintermute, J. Z. Joseph Xu, and J. E. Laird, "Sorts: A human-level approach to real-time strategy AI," in *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*, 2007, pp. 55–60.

[14] M. Cutumisu and D. Szafron, "An architecture for game behavior AI: Behavior multi-queues," in *AI and Interactive Digital Entertainment Conference, AIIDE (AAAI)*, 2009.

[15] D. Isla, "Handling complexity in the Halo 2 AI," in *Game Developers Conference*, 2005.

[16] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction (Adaptive Computation and Machine Learning)*. The MIT Press, March 1998.

[17] S. Ontañón, K. Mishra, N. Sugandh, and A. Ram, "Case-based planning and execution for real-time strategy games," in *ICCBR '07*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 164–178.

[18] M. Chung, M. Buro, and J. Schaeffer, "Monte carlo planning in rts games," in *IEEE Symposium on Computational Intelligence and Games (CIG)*, 2005.

[19] J. Diard, P. Bessière, and E. Mazer, "A survey of probabilistic models using the bayesian programming methodology as a unifying framework," in *Conference on Computational Intelligence, Robotics and Autonomous Systems, CIRAS*, 2003.

[20] D. J. C. MacKay, *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, 2003.

[21] M. J. Beal, "Variational algorithms for approximate bayesian inference," *PhD. Thesis*, 2003.

[22] P. Bessière, C. Laugier, and R. Siegwart, *Probabilistic Reasoning and Decision Making in Sensory-Motor Systems*, 1st ed. Springer Publishing Company, Incorporated, 2008.

[23] F. Colas, J. Diard, and P. Bessire, "Common bayesian models for common cognitive issues," *Acta Biotheoretica*, vol. 58, pp. 191–216, 2010.

[24] R. Le Hy, A. Arrigoni, P. Bessiere, and O. Lebeltel, "Teaching bayesian behaviours to video game characters," *Robotics and Autonomous Systems*, vol. 47, pp. 177–185, 2004.

[25] J. Asmuth, L. Li, M. Littman, A. Nouri, and D. Wingate, "A bayesian sampling approach to exploration in reinforcement learning," in *Uncertainty in Artificial Intelligence, UAI*. AUAI Press, 2009, pp. 19–26.