UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Informatique**

Arrêté ministérial : 7 août 2006

Présentée par

Gabriel Synnaeve

Thèse dirigée par **Pierre Bessière**

préparée au sein **Laboratoire d'Informatique de Grenoble**
et de **École Doctorale de Mathématiques, Sciences et Technologies de l'Information, Informatique**

Bayesian Programming Applied to Multi-Player Video Games

Thèse soutenue publiquement le **XXX**,
devant le jury composé de :

...Civilité, Prénom et Nom...
...titre et affiliation..., Président
...Civilité, Prénom et Nom...
...titre et affiliation..., Rapporteur
...Civilité, Prénom et Nom...
...titre et affiliation..., Rapporteur
...Civilité, Prénom et Nom...
...titre et affiliation..., Examinateur
...Civilité, Prénom et Nom...
...titre et affiliation..., Examinateur
...Civilité, Prénom et Nom...
...titre et affiliation..., Examinatrice
...Civilité, Prénom et Nom...
...titre et affiliation..., Directeur de thèse
...Civilité, Prénom et Nom...
...titre et affiliation..., Invité
...Civilité, Prénom et Nom...
...titre et affiliation..., Invitée

Contents

Chapter 1

Introduction

1.1 Contributions

Decentralization

Learnings

Hierarchy

1.2 Reading Map

À la MacKay (Industry vs Research)

Chapter 2

Game AI

David: What is the primary goal?
Joshua: You should know, Professor. You programmed me.
David: Oh, come on. What is the primary goal?
Joshua: To win the game.
Wargames (1983)

Or is it? "Game AI", simultaneously a research topic, an industry standard practice, from a staple to a part of the gameplay. Its uses range from character animation, to behavior modeling and strategic play. In this chapter, we will give our educated guess about the goals of game AI, and review what exists for a broad category of games: single player games, abstract strategy games, partial information and/or stochastic games, computer games. Let us then focus on game-play (from a player point of view) characteristics of theses games so that we can enumerate game AI needs.

2.1 Goals of Game AI

Non-playing characters (NPC), also called "mobs", are here to stay. Being it in ever more immersive single player adventures (The Elder Scrolls V: Skyrim), part of a cooperative gameplay (World of Warcraft, Left 4 Dead) or as helpers on our side or trainers against us ("pets", strategy games), they are of interest for the game industry, but also for robotics, to study human cognition and for artificial intelligence in the large.

Win

During the last decade, the video games industry has seen the emergence of "e-sport". It is the professionalization of specific competitive games at the higher levels, as in sports: with spectators, leagues, sponsors, fans and broadcasts. A list of major electronic sports games includes (but is not limited to): StarCraft: Brood War, Counter-Strike, Quake III, Warcraft III, Halo, StarCraft II. The first game to have had pro-gamers* was StarCraft: Brood War*, in Korea, with top players earning more than Korean top soccer players. Top players earn more than $400,000 a year but the professional average is below, around $50-60,000 a year [Contracts, 2007], against the average South Korean salary at $16,300 in 2010. Currently, Brood War is being slowly phased out to StarCraft II (still 4.9 millions players in South Korea in 2011 [CitationNeeded, 0000]). There are TV channels broadcasting Brood War (OnGameNet, previously also MBC Game) or StarCraft II (GOM TV, streaming) and for which it constitutes a major chunk of the air time. "E-sport" is important to the subject of game AI because it ensures competitiveness of the human players. It is less challenging to write a competitive AI for game played by few and without competitions than to write an AI for Chess, Go or StarCraft. E-sport, through the distribution of

"replays*" also ensures a constant and heavy flow of human player data to mine and learn from. Finally, cognitive science researchers (like the Simon Fraser University Cognitive Science Lab) study the cognitive aspects (attention, learning) of high level RTS playing[CitationNeeded, 0000].

Good human players, through their ability to learn and adapt, and through high-level strategic reasoning, are still undefeated. Single players are often frustrated by the NPC behaviors in non-linear (not fully scripted) games. Nowadays, video games AI could be used part of the gameplay as a challenge to the player. This is not the case in most of the games though, in decreasing order of resolution of the problem[1]: fast FPS* (first person shooters), team FPS, RPG* (role playing games), MMORPG* (Massively Multi-player Online RPG), RTS* (Real-Time Strategy). These games in which artificial intelligences do not beat top human players on equal footing requires increasingly more cheats to even be a challenge (not for long as they mostly do not adapt). AI cheats encompass (but are not limited to):

- RPG NPC often have at least 10 times more hit points (health points) than their human counterparts in equal numbers,
- FPS bots can see through walls and use perfect aiming,
- RTS bots see through the "fog of war" and have free additional resources.

How do we build game robotic players ("bots", AI, NPC) which can provide some challenge, or be helpful without being frustrating, while staying fun?

Fun

The main purpose of gaming is entertainment. Of course, there are subgenres of serious gaming or "gamification" of learning, but the majority of people playing games are having fun. Cheating AI are not fun, and so the re-playability of single player games is very low. The vast majority of games which are still played after the single player mode are multi-player games, because humans are the most fun to play with. So how do we get game AI to be fun to play with? The answer seems to be 3-fold:

- For competitive and PvP* (players versus players) games: improve game AI so that it can play well *on equal footing with humans*,
- for cooperative and PvE* (players vs environment) games: optimize the AI for fun, "epic wins": the empowerment of playing your best and just barely winning,
- give the AI all the tools to adapt the game to the players: AI directors* (as in Left 4 Dead* and Dark Spore*), procedural content generation (e.g. automatic personalized Mario [Shaker et al., 2010]).

In all cases, a good AI should be able to learn for the players' actions, recognize their behavior to deal with it in the most entertaining way. Examples for a few mainstream games: World of Warcraft instances or StarCraft II missions could be less predictable (less scripted) and always "just hard enough", Battlefield 3 or Call of Duty opponents could have a longer life expectancy (5 seconds in some cases), Skyrim's follower NPC could avoid blocking the player in doors, or going in front when she casts fireballs.

Programming

How do game developers want to deal with game AI programming? We have to understand the needs of industry game AI programmers:

[1]We deal in gameplay potentials here, particularly considering non-linear games, current RPG and MMORPG are often linearly limited *because* of the untracted"world interacting NPC" AI problem. Otherwise, linear RPG AI fare often better than team FPS AI.

- computational efficiency: most games are real-time systems, 3D graphics are computationally intensive, as a result the AI CPU budget is low,

- game designers often want to remain in control of the behaviors, so game AI programmers have to provide authoring tools,

- AI code has to scale with the state spaces while being debuggable,

- AI behaviors have to scale with the possible game states (which are not all predictable due to the presence of the human player),

- icing on the cake: re-use accross games (game independant logic).

So, programmers can "hard code" the behaviors and their switches, for some structuring they have finite state machines [Houlette and Fu, 2003]. This solution does not scale well (exponential increase in the number of transitions), nor do they generate autonomous behavior, and they can be cumbersome for the game designers to interact with. Hierarchical FSM [CitationNeeded, 0000] is a partial answer to these problems: they scale better due to the sharing of transitions between macro-states and are more readable for game designers who can zoom-in on macro/englobing states. They still represent way too much programming work for complex behavior and are not more autonomous than classic FSM. Planning (using a search heuristic in the states space) efficiently gives autonomy to virtual characters. Planners like hierarchical task networks (HTN [Erol et al., 1994], Armed Assault, Killzone 2 [van der Sterren, 2009, Champandard et al., 2009]) or STRIPS ([Fikes and Nilsson, 1971], F.E.A.R [Orkin, 2006]) generate complex behaviors in the space of the combinations of specified states, and the logic can be re-used accross games. The drawbacks can be a large computational budget (for many agents and/or a complex world), the difficulty to specify reactive behavior, and less (or harder) control from the game designers. Behavior trees (Halo 2 [Isla, 2005], Spore) are a popular in-between HTN and HFSM technique providing scability through a tree-like hierarchy, control through tree editing and some autonomy through a search heuristic (for valid nodes). A transversal technique for ease of use is to program game AI with a script (LUA, Python) or domain specific language (DSL*). From a programming or design point of view, it will have the drawbacks of the models it is based on. If everything is allowed (low-level inputs and outputs directly in the DSL), everything is possible at the cost of cumbersome programming, debugging and few re-use.

Even with scalable[2] architectures like behavior trees or the autonomy that planning provides, there are limitations (burdens on programmers/designers or CPU/GPU):

- complex worlds require either very long description of the state (in propositional logic) or high expressivity (higher order logics) to specify well-defined behaviors,

- the search space of possible actions increases exponentially with the interactivity (complexity) of the world, thus requiring ever more efficient pruning techniques,

- once human players are in the loop (is it not the purpose of a game?), uncertainty has to be taken into account. Previous approaches can be "patched" to deal with uncertainty, at what cost?

Our thesis is that we can learn complex behaviors from exploration or observations (of human players) without the need to be explicitely programmed. Furthermore, the game designers can stay in control by choosing which demonstration to learn from and tuning parameters by hand if wanted. Le Hy et al. [2004] showed it in the case of FPS AI (Unreal Tournament), with *inverse programming* to learn reactive behaviors from human demonstration. We extend it to tactical and even strategic behaviors.

[2]both computationally and in the number of lines of codes to write to produce a new behavior

2.2 Single Player Games

Single player games are not the main focus of our thesis, but they present a few interesting AI characteristics. They encompass all kinds of human cognitive abilities, from reflexes to higher level thinking.

Action games

Platform games (Mario, Sonic), time attack racing games (TrackMania), solo shoot-them-up ("schmups", Space Invaders, DodonPachi), sports games and rhythm games (Dance Dance Revolution, Guitar Hero) are games of reflexes, skill and level knowledge. The main components of game AI in these genres is a quick path search heuristic, often with a dynamic environment. There have been Mario [Togelius et al., 2010], PacMan [Rohlfshagen and Lucas, 2011] and racing competitions [Loiacono et al., 2008]. The winners often use (clever) heuristics coupled with a search algorithm (A* for instance). As there are no human opponents, reinforcement learning and genetic programming works well too.

XXX

Puzzles

Point and click (Monkey Island, Kyrandia, Day of the Tentacle), graphic adventure (Myst, Heavy Rain), (tile) puzzles (Minesweeper, Tetris) games are games of logical thinking and puzzle solving. The main components of game AI in these genres is an inference engine with sufficient domain knowledge (an ontology). AI research is not particularly active in the genre of puzzle games, perhaps because solving them has more to do with writing down the ontology than with using new AI techniques. A classic well-studied logic-based, combinatorial puzzle is Sudoku, which has been formulated as a SAT-solving [Lynce and Ouaknine, 2006] and constraint satisfaction problem [Simonis, 2005].

2.3 Abstract Strategy Games

Tic-tac-toe, Minimax

Tic-tac-toe (noughts and crosses) is a solved game*, meaning that it can be played optimally from each possible position. How did it came to get solved? Each and every possible positions (26,830) have been analyzed by a Minimax (or its variant Negamax) algorithm. Minimax is an algorithm which can be used to determine the optimal score a player can get for a move in a zero-sum game*. The Minimax theorem states:

Theorem. *For every two-person, zero-sum game with finitely many strategies, there exists a value V and a mixed strategy for each player, such that (a) Given player 2's strategy, the best payoff possible for player 1 is V, and (b) Given player 1's strategy, the best payoff possible for player 2 is -V.*

Applying this theorem to Tic-tac-toe, we can say that winning is +1 point for the player and losing is -1, while draw is 0. The exhaustive search algorithm which takes this property into account in described in Algorithm 1. The result of applying this algorithm to the Tic-tac-toe situation of Fig. 2.1 is exhaustively represented in Fig. 2.2. For zero-sum games ("strictly competitive games" as abstract strategy games discussed here), there is a (simpler) Minimax variant called Negamax, shown in Algorithm 5 in Appendix A.

Checkers, Alpha-beta

Checkers, Chess and Go are also zero sum, perfect-information*, partisan*, deterministic strategy game. Theoretically, they all can be solved by exhaustive Minimax. In practice though, it is often intractable: their bounded versions are at least in PSPACE and their unbounded versions are EXPTIME-hard [Hearn and Demaine, 2009]. The state space complexity of Checkers is the smallest of the 3 above-mentioned

Algorithm 1 Minimax algorithm

```
function MINI(depth)
    if depth ≤ 0 then
        return −value()
    end if
    min ← +∞
    for all possible moves do
        score ← maxi(depth − 1)
        if score < min then
            min ← score
        end if
    end for
    return min
end function
function MAXI(depth)
    if depth ≤ 0 then
        return value()
    end if
    max ← −∞
    for all possible moves do
        score ← mini(depth − 1)
        if score > min then
            max ← score
        end if
    end for
    return max
end function
```

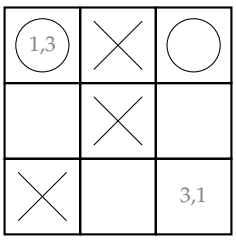


Figure 2.1: A Tic-tac-toe board position, "circles" turn to play. The couples of numbers explain the numbering (left to right, bottom to top, starting at 1) of the grid.

games with $\approxeq 5.10^{20}$ possible positions, and as a matter of fact, Checkers have been solved completely[Schaeffer et al., 2007]. We can see the complexity of Minimax as $O(b^d)$ with b the average branching factor of the tree (to search) and d the average length (depth) of the game. For Checkers $b \approxeq 10$, while the mean game length is 70 [Allis, 1994]. It is already too large to have been solved by Minimax alone (on current hardware). From 1989 to 2007, there were artificial intelligences competitions on Checkers, all using at least alpha-beta pruning: a technique to make efficient cuts in the Minimax search tree while not losing optimality.

Alpha-beta pruning (see Algorithm 2) is a branch-and-bound algorithm which (if the best nodes are searched first) can reduce Minimax to a $O(b^{d/2}) = O(\sqrt{b^d})$ complexity ($O(b^{3d/4})$ for a random ordering of nodes). α is the maximum score than we (the maximizing player) are assured to get given what we already evaluated, while β is the minimum score than the minimizing player is assured to get. When evaluating more and more nodes, we can only get a better estimate and so α can only increase while β can only decrease. If we find a node for which β becomes less than α, it means that this position

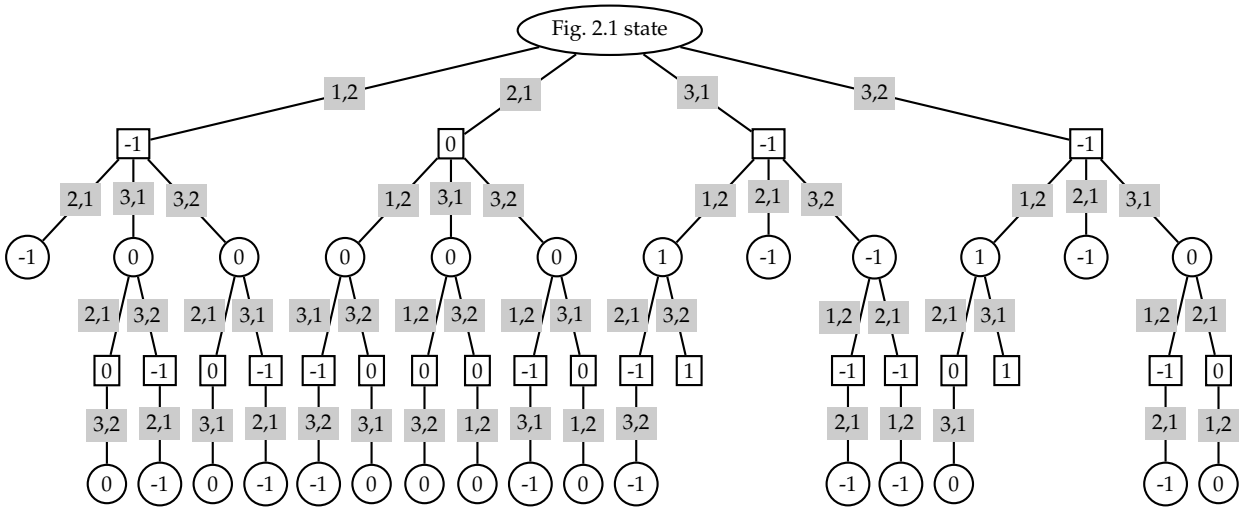


Figure 2.2: Minimax tree with initial position at Fig. 2.1 state, **nodes** are states and **edges** are transitions, labeled with the move. Leafs are end-game states: 1 point for the win and -1 for the loss. Player is "circles" and plays first (first edges are player's moves).

Algorithm 2 Alpha-beta algorithm

```
function ALPHABETA(node,depth,α,β,player)
    if depth ≤ 0 then
        return value(player)
    end if
    if player == us then
        for all possible moves do
            α ← max(α, alphabeta(child, depth − 1, α, β, opponent))
            if β ≤ α then
                break
            end if
        end for
        return α
    else
        for all possible moves do
            β ← min(β, alphabeta(child, depth − 1, α, β, us))
            if β ≤ α then
                break
            end if
        end for
        return β
    end if
end function
```

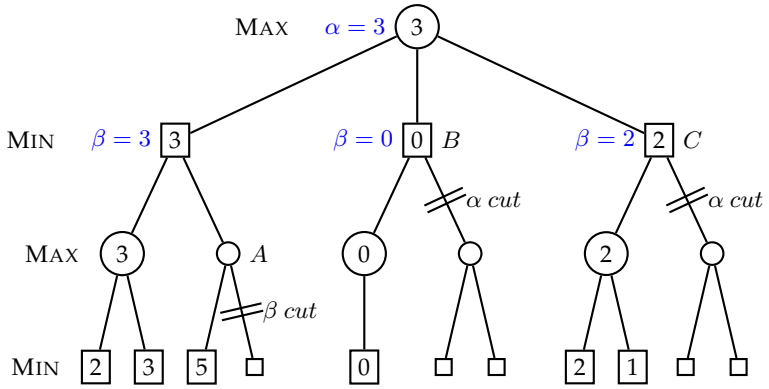


Figure 2.3: Alpha-beta cuts on a Minimax tree, **nodes** are states and **edges** are transitions, labeled with the values of positions at the bottom (max depth). Here is the trace of the algorithm: **1.** descend leftmost first and evaluated 2 and 3, **2.** percolate max(2,3) higher up to set $\beta = 3$, **3.** β-cut in A because its value is at least 5 (which is superior to $\beta = 3$), **4.** Update of $\alpha = 3$ at the top node, **5.** α-cut in B because it is at most 0 (which is inferior to $\alpha = 3$), **6.** α-cut in C because it is at most 2, **7.** conclude the best value for the top node is 3.

results from sub-optimal play. When it is because of an update of β, the sub-optimal play is on the side of the maximizing player (his α is not high enough to be optimal and/or the minimizing player has a winning move faster in the current sub-tree) and this situation is called an α cut-off. On the contrary, when the cut results from an update of α, it is called a β cut-off and means that the minimizing player would have to play sub-optimally to get into this sub-tree. When Starting the game, α is initialized to $-\infty$ and β to $+\infty$. A worked example is given on Figure 2.3. Prior to Checkers being solved (meaning that we have a database of which moves to play for all positions resulting from optimal play), or if playing against a sub-optimal opponent, Alpha-beta is going to be helpful to search much deeper than Minimax in the same allowed time. The best Checkers program (since the 90s), which is also the project which solved Checkers [Schaeffer et al., 2007], Chinook, has openings and end-game (for lest than eight pieces of fewer) books, and for the mid-game (when there are more possible moves) relies on a deep search algorithm. So, appart for the beginning and the ending of the game, for which it plays by looking up a database, it used a search algorithm. As Minimax and Alpha-beta are depth first search heuristics, all programs which have to answer in a fixed limit of time use *iterative deepening*, a technique which consists in fixing limited depth which will be considered maximal and evaluating this position. As it does not relies in winning moves at the bottom (remember, the search space is too big in $branching^{depth}$), we need moves evaluation heuristics. We then iterate on growing the maximal depth for which we consider moves, but we are at least sure to have a move to play in a short time (at least the greedy depth 1 found move).

Chess, Heuristics

With a branching factor of $\approxeq 35$ and an average game length of 80 moves [Shannon, 1950], the average game-tree complexity of chess is $35^{80} \approxeq 3.10^{123}$. Shannon [1950] also estimated the number of possible positions (Shannon number) to be of the order of 10^{43}. Chess AI needed a little more than just Alpha-beta to win against top human players (not that Checkers could not benefit it!), particularly on 1996 hardware (first win of a computer against a reigning world champion, Deep Blue on Garry Kasparov). Once an AI has openings and ending books (databases to look-up for classical moves), how can we search deeper during the game, or how can we evaluate better a situation? In iterative deepening Alpha-beta (or other search algorithms like Negascout [Reinefeld, 1983] or MTD-f[Plaat, 1996]), one needs to know the value of a move at the maximal depth. If it does not correspond to the end of the

game, there is a need for a evaluation heuristic. Some may be straight forward, like the resulting value of an exchange in pieces points. But some strategies sacrifice a queen in favor of a more advantageous tactical position or a checkmate, so evaluation heuristics need to take tactical positions into account. In Deep Blue, the evaluation function had 8000 cases, with 4000 positions in the openings book, all learned from 700,000 grandmaster games [Campbell et al., 2002]. Nowadays, Chess programs are better than deep blue and generally also search less positions. For instance, Pocket Fritz (HIARCS engine) beats current grandmasters [CitationNeeded, 0000] while evaluating 20,000 positions per second (740 MIPS on a smartphone) against Deep Blue's (11.38 GFlops) 200 millions per second.

Go, Monte-Carlo Tree Search

With an estimated number of legal 19x19 Go positions of $2.081681994 * 10^{170}$ Tromp and Farnebäck [2006], and an average branching factor above Chess for gobans* above 9x9, Go sets another limit for AI. MoGo [Gelly and Wang, 2006, Gelly et al., 2006] introduced Upper Confidence Bounds for Trees (UCT*) for Monte-Carlo Tree Search (MCTS*) in Go AI successfully (being the best 9x9 and 13x13 Go program and the first to win against a pro on 9x9). The approach of MCTS is to randomly sample in the search tree (which is too big to be searched entirely), instead of systematically expanding the build tree as in Minimax. For that, all we need is to have, for each node $node$ in the search tree, $Q(node)$ the sum of the simulations rewards on all the runs through $node$, and $N(node)$ the visits count of $node$. Algorithm 3 details the MCTS algorithm and Fig. 2.4 explains the principle.

Algorithm 3 Monte-Carlo Tree Search algorithm. EXPANDFROM($node$) is the tree (growing) policy function on how to select where to search from situation $node$ (exploration or exploitation?) and how to expand the game tree (deep-first, breadth-first, heuristics?) in case of untried actions. EVALUATE($tree$) may have 2 behaviors: **1.** if $tree$ is complete (terminal), it gives an evaluation according to games rules, **2.** if $tree$ is incomplete, it has to give an estimation, either through simulation (for instance play at random) or an heuristic. BESTCHILD picks the action that leads to the better value/reward from $node$. MERGE($node, tree$) changes the existing tree (with $node$) to take all the $Q(\nu) \forall \nu \in tree$ (new) values into account. If $tree$ contains new nodes (there were some exploration), they are added to $node$ at the right positions.

```
function MCTS(node)
    while computational time left do
        tree ← EXPANDFROM(node)
        tree.values ← EVALUATE(tree)
        MERGE(node, tree)
    end while
    return BESTCHILD(node)
end function
```

Algorithm 4 UCB1 EXPANDFROM

```
function EXPANDFROM(node)
    if node is terminal then
        return node    // terminal
    end if
    if ∃c ∈ node.children s.t. N(c) = 0 then
        return c    // grow
    end if
    return EXPANDFROM(arg max_{c∈node.children)} Q(c)/N(c) + k√(ln N(node)/N(c)))    // select
end function
```

UCT specializes MCTS in that it specifies EXPANDFROM (as in Algorithm. 4) tree policy with a specific exploration-exploitation trade-off. UCB1 [Kocsis and Szepesvári, 2006] views the tree policy as

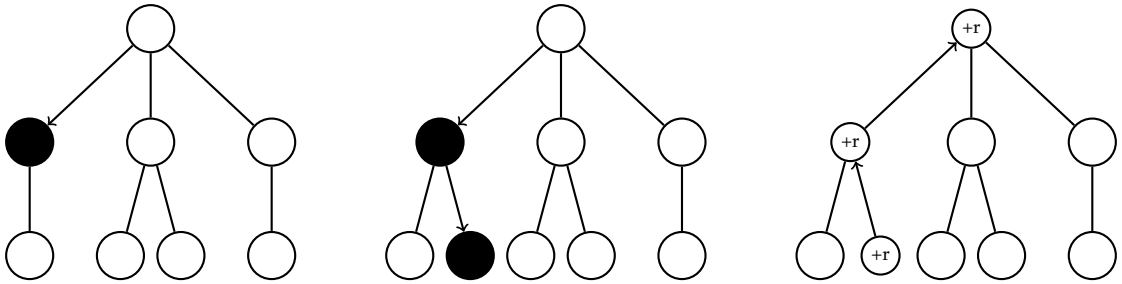


Figure 2.4: An iteration of the **while** loop in MCTS, from left to right: EXPANDFROM select, EXPANDFROM grow, EVALUATE & MERGE.

a multi-armed bandit problem and so EXPANDFROM($node$) UCB1 chooses the arms with the best upper confidence bound:

$$\underset{c \in node.children}{\arg\max} \frac{Q(c)}{N(c)} + k\sqrt{\frac{\ln N(node)}{N(c)}}$$

in which k fixes the exploration-exploitation trade-off: $\frac{Q(c)}{N(c)}$ is simply the average reward when going through c so we have exploitation only for $k = 0$ and exploration only for $k = \infty$.

Kocsis and Szepesvári [2006] showed that the probability of selecting sub-optimal actions converges to zero and so that UCT MCTS converges to the minimax tree and so is optimal. Empirically, they found several convergences rates of UCT to be in $O(b^{d/2})$, as fast as Alpha-beta tree search, and able to deal with larger problems (with some error). For a broader survey on MCTS methods, see [Browne et al., 2012].

With Go, we see clearly that humans do not play abstract strategy games using the same approach. Top Go players can reason about their opponent's move, but they seem to be able to do it in a qualitative manner, at another scale. So, while tree search algorithms help a lot for tactical play in Go, particularly by integrating openings/ending knowledge, pattern macthing algorithms are not yet at the strategical level of human players. When a MCTS algorithm learns something, it stays at the level of possible actions (even considering positional hashing*), while the human player seems to be able to generalize, and re-use heuristics learned at another level.

2.4 Games with Uncertainty

An exhaustive list of games or even of games genres is beyond the scope/range of this thesis. All uncertainty boils down to incompleteness of information, being it the physics of the dice being thrown or the inability to measure what is happening in the opponent's brain. However, we will speak of 2 types (sources) of uncertainty: extensional uncertainty, which is due to incompleteness in direct, measurable information, and intentional uncertainty, which is related to randomness in the game or in (the opponent's) decisions. The purest extensional uncertainty being acting without sensing while the purest intentional uncertainty would be for our acts to be the result of a perfect random generator. The uncertainty coming from the opponent's mind/cognition lies in between, depending on the simplicity to model the game as an optimization procedure. The harder the game is to model, the harder it is to model the trains of thoughts our opponents can follow.

Monopoly

In Monopoly, there is few hidden information (*Chance* and *Community Chest* cards only), but there is randomness in the throwing of dice[3], and a substantial influence of skill (player's decision). A very basic playing strategy would be to just look at the ruturn on investment (ROI) with regard to prices, rents and frequencies, choosing only based on the money you have and the possible actions of buying or not. A less naive way to play should evaluate the questions of buying with regard to what we already own, what others own, our cash and advancement in the game. The complete state space is huge (places for each players $\times$ their money $\times$ their possessions), but according to Ash and Bishop [1972], we can model the game for one player (as he has no influence on the dice rolls and decisions of others) as a Markov process on 120 ordered pairs: 40 board spaces $\times$ possible number of doubles rolled so far in this turn (0, 1, 2). With this model, it is possible to compute more than simple ROI and derive applicable and interesting strategies. So, even in monopoly, which is not lottery playing or simple dice throwing, a simple probabilistic modeling yields a robust strategy. Additionally, Frayn [2005] used genetic algorithms to generate the most efficient strategies for portfolio management.

Monopoly is an example of a game in which we have complete direct information about the state of the game, intentional uncertainty due to the roll of the dice (randomness) can be dealt with thanks to probabilistic modeling (Markov processes here). The opponent's actions are relatively easy to model due to the fact that the goal is to maximize cash and that there are not many different efficient strategies (not many Nash equilibrium if it were a stricter game) to attain it. In general, the presence of chance does not invalidates previous (game theoretic / game trees) approaches but transforms exact computational techniques into stochastic ones: finite states machines become probabilistic Bayesian networks for instance.

Battleship

Battleship (also called "naval combat") is a guessing game generally played with two 10x10 grids for each players: one is the player's ships grid, and one is to remember/infer the opponent's ships positions. The goal is to guess where the enemy ships are and sink them by firing shots (torpedoes). There is **incompleteness** of information but no randomness. Incompleteness can be dealt with with probabilistic reasoning. The classic setup of the game consist in two ships of length 3 and one ship of each lengths of 2, 4 and 5; in this setup, there are 1,925,751,392 possible arrangements for the ships. The way to take advantage of all possible information is to update the probability that there is a ship for all the squares each time we have additional information. So for the 10x10 grid we have a 10x10 matrix $O_{1:10,1:10}$ with $O_{i,j} \in \{true, false\}$ being the ith row and jth column random variable of the case being occupied. With $ships$ being unsunk ships, we always have

$$\sum_{i,j} P(O_{i,j} = true) = \frac{\sum_{k \in ships} length(k)}{10 \times 10}$$

. For instance for a ship of size 3 alone at the beginning we can have prior distributions on $O_{1:10,1:10}$ by looking at combinatorics of its placements (see Fig. 2.5). We can also have priors on where the opponent likes to place her ships. Then each round we will either hit or miss in i, j. When we hit, we know $P(O_{i,j} = true) = 1.0$ and will have to revise the probabilities of surrounding areas, and everywhere if we learned the size of the ship, with possible placement of ships. If we did not sunk a ship, the probabilities of uncertain (not 0.0 or 1.0) positions around i, j will be highered according to the sizes of remaining ships. If we miss, we know $P(O_{i,j}) = false$ and can also revise (lower) the surrounding probabilities, an example of that effect is shown in Fig. 2.5.

Battleship is a game with few intensional uncertainty (no randomness), particularly because the goal quite strongly conditions the action (sink ships as fast as possible) but a large part of extensional

[3]Note that the sum of two uniform distributions is not a uniform but a Irwin-Hall, $\forall n > 1, P([\sum_{i=1}^{n} (X_i \in U(0,1))] = x) \propto \frac{1}{(n-1)!} \sum_{k=0}^{n} (-1)^k \binom{n}{k} \max((x-k)^{n-1}, 0)$, converging towards a Gaussian (central limit theorem).

uncertainty (incompleteness of direct information), which goes down rather quickly once we act, if we update a probabilistic model of the map/grid. If we compare Battleship to a variant in which we could see the adversary board, playing would be straightforward (just hit ships we know the position on the board), now in real Battleship we have to model our uncertainty due to the incompleteness of information, without even beginning to take into account the psychology of the opponent in placement as a prior. The cost of solving an imperfect information game increases greatly from its perfect information variant: it seems to be easier to model stochasticity (chance, a source of randomness) than to model a hidden (complex) system for which we only observe (indirect) effects.

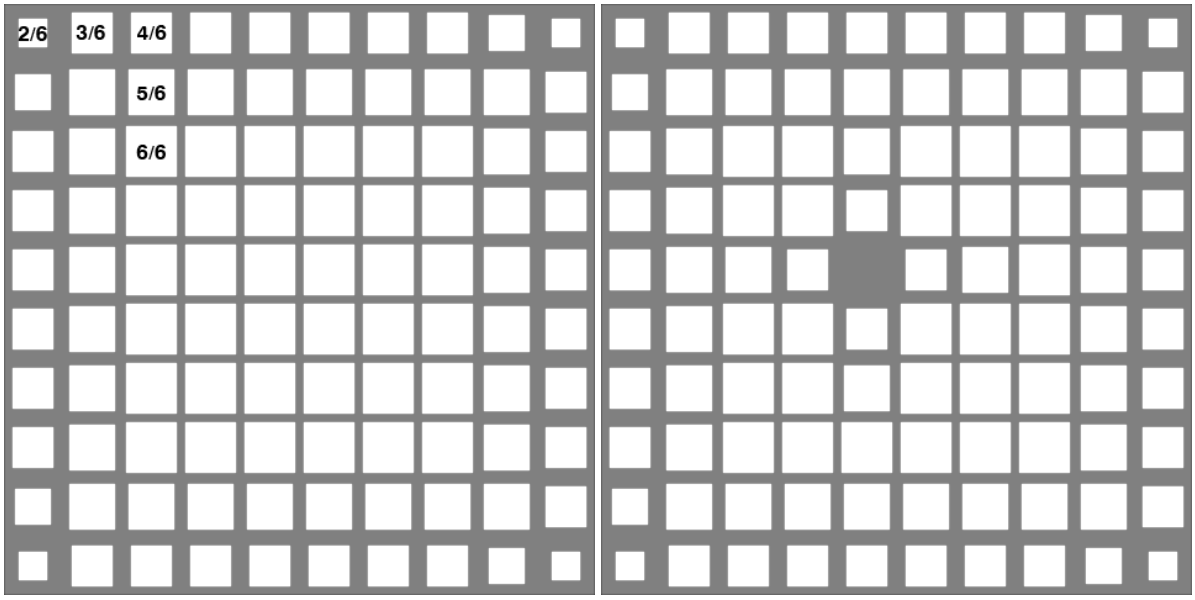


Figure 2.5: Left: visualization of probabilities for squares to contain a ship of size 3 ($P(O_{i,j}) = true$) initially, assuming uniform distribution of this type of ship. Annotations correspond to the *number of combinations* (on six, the maximum number of conformations), Right: same probability after a miss in (5, 5). The larger the white area, the higher the probability.

Poker

Poker[4] is a zero-sum (without the house's cut), imperfect information and stochastic betting game. Poker "AI" is as old as game theory [Nash, 1951], but the research effort for human-level Poker AI started in the end of the 90s. The interest for Poker AI is such that there are annual AAAI computer Poker competitions[5]. Billings et al. [1998] defend Poker as an interesting game for decision-making research, because the task of building a good/high level Poker AI (player) entails to take decisions with incomplete information about the state of the game, incomplete information about the opponents' intentions, and model their thoughts process to be able to bluff efficiently. A Bayesian network can combine these uncertainties and represent the player's hand, the opponents' hands and their playing behavior conditioned upon the hand, as in [Korb et al., 1999]. A simple "risk evaluating" AI (folding and raising according to the outcomes of its hands) will not prevail against good human players. Bluffing, as described by Von Neumann and Morgenstern [1944] "to create uncertainty in the opponent's mind", is an element of Poker which needs its own modeling. Southey et al. [2005] also give a Bayesian treatment to Poker, separating the uncertainty resulting from the game (draw of cards) and from the opponents' strategies, and focusing on bluff. From a game theoretic point of view, Poker is a *Bayesian*

[4]We deal mainly with the *Limit Hold'em* variant of Poker.
[5]`http://www.computerpokercompetition.org/`

game, which needs extensive form modeling (possible game trees along with the agents' information state). Koller and Pfeffer [1997] used the sequence form transformation, the set of realization weights of the sequences of moves, to search over the space of randomized strategies for Bayesian games automatically. Unfortunately, strict game theoretic optimal strategies for full-scale Poker are not tractable this way, two players Texas Hold'em having a state space $\approxeq O(10^{18})$. Billings et al. [2003] approximated the game-theoretic optimal strategies through abstraction and are able to beat strong human players (not world-class opponents).

Poker is a game with both extensional and intentional uncertainty, from the fact that the opponents' hands are hidden, the chance in the draw of the cards, the opponents' model about the game state and their model about our mental state(s) (leading our decision(s)). While the iterated reasoning ("if she does A, I can do B") is (theoretically) finite in Chess due to perfect information, it is not the case in Poker ("I think she thinks I think..."). The combination of different sources of uncertainty (as in Poker) makes it complex to deal with it (somehow, the sources of uncertainty must be separated), and we will see that both these sources of uncertainties arise (at different levels) in video games.

2.5 FPS

Gameplay and AI

First person shooters gameplay* consist in controlling an agent in first person view, centered on the weapon, a gun for instance. The firsts FPS popular enough to bring the genre its name were Wolfeinstein 3D and Doom, by iD Software. Other classic FPS (series) include Duke Nukem 3D, Quake, Half-Life, Team Fortress, Counter-Strike, Unreal Tournament, Tribes, Halo, Medal of Honor, Call of Duty, Battlefield, etc. The distinction between "fast FPS" (e.g. Quake series, Unreal Tournament series) and others is made on the speed at which the player moves. In "fast FPS", the player is always jumping, running much faster and playing more in 3 dimensions than on discretely separate 2D ground planes. Game types include (but are not limited to):

- single-player missions, depending of the game design.
- capture-the-flag (CTF), in which a player has to take the flag inside the enemy camp and bring it back in her own base.
- free-for-all (FFA), in which there are no alliances.
- team deathmatch (TD), in which two (or more) teams fight on score.
- various gather and escort (including hostage or payload modes), in which one team has to find and escort something/somebody to another location.
- duel/tournament/deathmatch, 1 vs 1 matches (mainly "fast FPS").

From these various game types, the player has to maximize its damage (or positive actions) output while staying alive. For that, she will navigate her avatar in an uncertain environment (partial information and other players intentions) and shoot (or not) at targets with specific weapons.

Some games allow for instant (or delayed, but asynchronous to other players) respawn, most likely in the "fast FPS" (Quake-like) games, while in others, the player has to wait for the end of the round to respawn. In some games, weapons, ammunitions, health, armor and items can be picked on the ground (mainly "fast FPS"), in others, they are fixed at the start or can be bought in game (with points). The map design can make the gameplay vary a lot, between indoors, outdoors, arena-like or linear maps. According to maps and gameplay styles, combat may be well-prepared with ambushes, sniping, indirect (zone damages), or close proximity (even to fist weapons). Most often, there are strong tactical positions and effective ways to attack them.

While "skill" (speed of the movements, accuracy of the shots) is easy to emulate for an AI, team-play is much harder for bots and it is always a key ability. Team-play is the combination of distributed evaluation of the situation, planning and distribution of specialized tasks. Very high skill also requires integrating over enemy's tactical plans and positions to be able to take indirect shots (grenades for instance) or better positioning (coming from their back), which is hard for AI too. An example of that is that very good human players consistently beat the best bots (nearing 100% accuracy) in Quake III (which is an almost pure skill "fast FPS"), because they take advantage of being seen just when their weapons reload or come from their back. Finally, bots which equal the humans by a higher accuracy are less fun to play with: it is a frustrating experience to be shot across the map, by a bot which was stuck in the door because it was pushed out of its trail.

State of the art

FPS AI consists in controlling an agent in a complex world: it can have to walk, run, crouch, jump, swim, interact with the environment and tools, and sometimes even fly. Additionally, it has to shoot at moving, coordinated and dangerous, targets. On a higher level, it may have to gather weapons, items or power-ups (health, armor, etc.), find interesting tactical locations, attack, chase, retreat...

The Quake III AI is a standard for Quake-like games [van Waveren and Rothkrantz, 2002]. It consists in a layered architecture (hierarchical) FSM. At the sensory-motor level, it has an Area Awareness System (AAS) which detects collisions, accessibility and computes paths. The level above provides intelligence for chat, goals (locations to go to), goals and weapons selection through fuzzy logic. Higher up, there are the behavior FSM ("seek goals", chase, retreat, fight, stand...) and production rules (if-then-else) for squad/team AI and orders. A team of bots always behaves following the orders of one of the bots. Bots have characters, which can be accessed/felt by human players, specified by fuzzy relations on "how much the bot wants to do, have, or use something". A genetic algorithm was used to optimize the fuzzy logic parameters for specific purposes (like performance). This bot is fun to play against and is considered a success, however Quake-like games makes it easy to have high level bots because very good players have high accuracy (no fire spreading), so they do not feel cheated if bots have a high accuracy too. Also, the game is mainly indoors, which facilitates tactics and terrain reasoning. Finally, cooperative behaviors are not very evolved and consist in acting together towards a goal, not with specialized behaviors for each agent.

The more recent FPS games have dealt with these limitations by using combinations of STRIPS planning (F.E.A.R. [Orkin, 2006]), hierarchical task networks (HTN) (Killzone 2 [Champandard et al., 2009] and ArmA [van der Sterren, 2009]), Behavior trees (Halo 2 [Isla, 2005]). Left4Dead (a cooperative PvE FPS) uses a global supervisor of the AI to set the pace of the threat to be the most enjoyable for the player [Booth, 2009].explain Behavior trees?
http://files.aigamedev.com/insiders/BehaviorTrees_Slides.pdf
http://altdevblogaday.com/2011/02/24/introduction-to-behavior-trees/
http://aigamedev.com/open/article/anatomy-ai-architecture/
http://www.cgf-ai.com/docs/straatman_remco_killzone_ai.pdf

In research, Laird [2001] focused on learning rules for opponent modeling, planning and reactive planning (on Quake), so that the robot builds plan by anticipating the opponent's actions. Le Hy et al. [2004] used robotics inspired Bayesian models to quickly learn the parameters from human players (on Unreal Tournament). Zanetti and Rhalibi [2004] and Westra and Dignum [2009] applied evolutionary neural networks to optimize Quake III bots. Predicting opponents positions is a central task to believability and has been solved successfully using particle filters [Bererton, 2004] and other models (like Hidden Semi-Markov Models) [Hladky and Bulitko, 2008]. Multi-objective neuro-evolution [Zanetti and Rhalibi, 2004, Schrum et al., 2011] combines learning from human traces with evolutionary learning for the structure of the artificial neural network, and leads to realistic (human-like) behaviors, in the context of the BotPrize challenge (judged by humans) [Hingston, 2009].

Challenges

Single-player FPS campaigns immersion could benefit from more realistic bots and clever squad tactics. Multi-player FPS are competitive games, and a better game AI should focus on:

- **believability** requires the AI to take decisions with the same informations than the human player (fairness) and to be able to deal with unknown situation.
- **surprise** and unpredictability is key for both performance and the long-term interest in the human player in the AI.
- **performance**, to give a challenge to human players, can be achieved through cooperative, planned and specialized behaviors.

2.6 (MMO)RPG

Gameplay and AI

Inspired directly by tabletop and live action role-playing games (Dungeon & Dragons) as new tools for the game masters, it is quite natural for the RPG to have ended up on computers. The firsts digital RPG were text (Wumpus) or ASCII-art (Rogue, NetHack) based. The gameplay evolved considerably with the technique. Nowadays, what we will call a role playing game (RPG) consist in the incarnation by the human player of an avatar (or a team of avatars) with a class: warrior, wizard, rogue, priest, etc., having different skills, spells, items, health points, stamina/energy/mana (magic energy) points. Generally, the story brings the player to solve puzzles and fight. In a fight, the player has to take decisions about what to do but plays a lesser role in performing the action than in a FPS game. In a FPS, she has to move the character (egocentrically) and aim to shoot; in a RPG, she has to position itself (often way less precisely and continually) and just decide which ability to use on which target (or a little more for "action RPG"). Classic RPG include: Fallout, The Elders Scrolls (from Arena to Skyrim), Secret of Mana, Zelda, Final Fantasy, Diablo, Baldur's Gate. A MMORPG (e.g. World of Warcraft, AION or EVE Online) consist in a role-playing game in a persistent, multi-player world. There usually are players-run factions fighting each others' (PvP) and players versus environment (PvE) situations. PvE is a cooperative task in which human players fight together against different NPC, and in which the cooperation is at the center of the gameplay. PvP is also a cooperative task, but more policy and reactions-based than a trained and learned choregraphy as for PvE. We can distinguish three types (or modality) of NPC which have different game AI needs:

- world/neutral/civilian NPC: gives quests, takes part in the world's or game's story, talks,
- "mob"/hostile NPC that the player will fight,
- "pets"/allied NPC: acts by the players' sides.

While immobile neutral NPC are bad, acting strangely is sometimes worse for the player's immersion. It is more fun for the player to battle with hostile NPC which are not too dumb or predictable. Players really expect allied NPC to at least not hinder them, and it is even better when they adapt to what the player is doing. Finally, for MMORPG, persistent character AI could maintain the player's avatar in the world when she is enjoying real-life.

State of the art

Methods used in FPS are also used in RPG. The needs are sometimes a little different for RPG games, for instance to have interruptible and/or modal behaviors, along with stronger story-telling capabilities [Kline, 2009, Riedl et al., 2011]. Behavior multi-queues [Cutumisu and Szafron, 2009] resolve the

problems of having resumable, collaborative, real-time and parallel behavior. Kline [2011] used an AI director to adapt the difficulty of Dark Spore to the performance of the player in real-time.

XXX

Challenges

There are mainly two axes for RPG games to bring more fun: interest in the game play(s), and immersion. For both these topics, we think game AI can bring a lot:

- **believability** of the agents will come from AI approaches than can deal with new situations, being it because they were not dealt with during game development (because the "possible situations" space is too big) or because they were brought by the players' unforeseeable actions. Scripts and strict policies approaches will be in difficulty here, and we will assist to other Skyrim's NPC blunders.
- **interest** (as opposed to boredom) for the human players in the game style of the AI will come from approaches which can generate different behaviors in a given situation. Expectable AI particularly affects replayability negatively.
- **performance** relative to the gameplay will come from AI approaches than can fully deal with cooperative behavior. One solution is to design mobs to be orders of magnitude stronger (in term of hit points and damages) than players characters, or more numerous. Another, arguably more entertaining, solution is to bring the mobs behavior to a point where they are a challenge for the team of human players.

Both believability and performance require to deal with uncertainty of the game environment. RPG AI problem spaces are not tractable for a frontal (low-level) search approach nor are there few enough situations to consider to just write a bunch of script and puppeteer artificial agents at any time.

2.7 RTS

As RTS are the central focus on this thesis, we will discuss specific problems and solution more in depth in their dedicated chapters, simply brushing here the underlying major research problems.

Gameplay and AI

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. Finally, at the actions/motor level, micro-management 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. Both [Laird and van Lent, 2001] and Gunn et al. [2009] propose that RTS AI is one of the most challenging genres, because all levels in the hierarchy of decisions are of importance.

In chronological order, RTS include (but are not limited to): Ancient Art of War, Herzog Zwei, Dune II, Warcraft, Command & Conquer, Warcraft II, C&C: Red Alert, Total Annihilation, Age of Empires,

StarCraft, Age of Empires II, Tzar, Cossacks, Homeworld, Battle Realms, Ground Control, Spring Engine games, Warcraft III, Total War, Warhammer 40k, Sins of a Solar Empire, Supreme Commander, StarCraft II.

XXX ???

State of the art

Buro [2004] called for AI research in RTS games and identified the technical challenges as adversarial planning under uncertainty, learning and opponent modeling, and spatial and temporal reasoning.

On planning under uncertainty, Aha et al. [2005] used case-based reasoning (CBR) to perform dynamic plan retrieval extracted from domain knowledge in Wargus (Warcraft II clone). Ontañón et al. [2007] based their real-time case-based planning (CBP) system on a plan dependency graph which is learned from human demonstration in Wargus. In [Mishra et al., 2008, Ontañón et al., 2010, Manish Meta, 2010], they used a knowledge-based approach to perform situation assessment to use the right plan, and revise it (integrating learning, planning, and problem solving in CBR), performing runtime adaptation by monitoring its performance. Trusty et al. [2008] used a genetic-algorithm inspired method to mix and optimize existing expert plan, also in Wargus. Chung et al. [2005] adapted Monte-Carlo tree search to planning in RTS games and applied it to a capture-the-flag mod of Open RTS. Balla and Fern [2009] applied UCT (MCTS algorithm) to tactical assault planning in Wargus. Reactive planning and goal-driven autonomy (that is, find the more relevant goal to follow in unknown situations) are studied in [Weber et al., 2010a,b]. Churchill and Buro [2011] used abstractions and heuristics to produce a real-time build-order planner.

On learning and opponent modeling, Schadd et al. [2007] used hierarchical classifiers to learn the opponent's behavior in Total Annihilation (Spring). Hsieh and Sun [2008] learned players' strategy models with CBR by mining replays of StarCraft. Weber and Mateas [2009] applied data-mining to StarCraft replays to learn to predict strategies. Hagelbäck and Johansson [2010] performed a study on what are the human like characteristics of play in RTS games. Kim et al. [2010] clusterized build-orders to learn them from replays. Kabanza et al. [2010] studied strategic and tactic plan and intent recognition by probabilistic weighting of plans from a plan library, on StarCraft. In [Synnaeve and Bessière, 2011b], we clusterized replays to annotate them with openings (strategies) and then learned the parameters of a predictive Bayesian model for strategies from StarCraft replays. In [Synnaeve and Bessière, 2011], we also learned parameters of a strategy prediction Bayesian model from StarCraft replays but in an unsupervised fashion (predicting build/tech trees). Dereszynski et al. [2011] also had an unsupervised learning approach by fitting HMM to players build actions (from StarCraft replays) and clustering the most probable HMM transitions to be subsets of strategies.

On spatial and temporal reasoning, Forbus et al. [2002] presented tactical qualitative description of terrain for wargames through geometric and pathfinding analysis. Perkins [2010] automatically extracted choke points and regions of StarCraft maps from a pruned Voronoi diagram. Southey et al. [2007] infered "complex agent motions from partial information" using hidden semi-markov models including A* as a motion model. Butler and Demiris [2010] applied forward and inverse models simulation to the prediction of units movements from partial observations onto a RTS. Weber et al. [2011] implemented a simpler particle filter for state estimation in StarCraft. Wintermute et al. [2007] used a cognitive approach mimicking human attention for tactics and units control. Miles et al. [2007] and Smith et al. [2010] co-evolved influence map trees for spatial (tactical) reasoning in RTS games. Danielsiek et al. [2008] combined flocking [Reynolds, 1987] with influence maps, while Preuss et al. [2010] enhanced it, supporting team composition and maneuvering by learning a self-organizing map. Hagelbäck and Johansson [2009] presented a multi-agent potential field based bot and we presented a similar unifying Bayesian model for micro-management [Synnaeve and Bessière, 2011a] in which units are attracted or repulsed by different real or virtual units or goals.

2.8 Games Characteristics

All the types of video games that we saw before require to deal with imperfect information and sometimes with randomness, while elaborating a strategy (possibly from underlying policies). From a game theoretic point of view, these video games are close to what is called a Bayesian game [Osborne and Rubinstein, 1994]. However, solving Bayesian games is non-trivial, there are no generic and efficient approaches, and so it has not been done formally for card games with more than a few cards. Billings et al. [2003] approximated a game theoretic solution for Poker through abstraction heuristics, it leads to believe than game theory can be applied at the higher (strategic) abstracted levels of video games.

We do not pretend to do a complete taxonomy of video games and AI (e.g. [Gunn et al., 2009]), but we wish to provide all the major informations to differentiate game genres (gameplays). To grasp the challenges they pose, we will provide abstract measures of complexity.

Combinatory

"How does the state of possible actions grow?" To measure this, we used a measure from perfect information zero-sum games (as Checkers, Chess and Go): the branching factor b and the depth n of a typical game. The complexity of a game (for taking a decision) is proportional to b^n. The average branching factor for a board game is easy to compute: it is the average number of possible moves for a given player. For Poker, we set $b = 3$ for *fold, check* and *raise*. n should then be defined over some time, the average number of events (decisions) per hour in Poker is between 20 to 240. For video-games, we defined b to be the average number of possible moves at each decision, so for "continuous" or "real-time" games it is some kind of function of the useful discretization of the virtual world at hand. n has to be defined as a frequency at which a player (artificial or not) has to take decisions to be competitive in the game, so we will give it in $n/time\_unit$. For instance, for a car (plane) racing game, $b \approxeq 50 - 500$ because b is a combination of throttle ($\ddot{x}$) and direction (θ) sampled values that are relevant for the game world, with n/min at least 60: a player needs to correct her trajectory *at least* once a second. In RTS games, $b \approxeq 200$ is a lower bound (in StarCraft we may have between 50 to 400 units to control), and very good amateurs and professional players perform more than 300 actions per minute.

The sheer size of b and n in video games make it seem intractable, but humans are able to play, and to play well. To explain this phenomenon, we introduce "vertical" and "horizontal" continuities in decision making. Fig. 2.6 shows how one can view the decision-making process in a video game: at different time scales, the player has to choose between strategies to follow, that can be realized with the help of different tactics. Finally, at the action/output/motor level, these tactics have to be implemented one way or the other. So, matching Fig. 2.6, we could design a Bayesian model:

- $S^{t,t-1} \in Attack, Defend, Seach, Hide$, the strategy variable
- $T^{t,t-1} \in Front, Back, Prepare, Rush$, the tactics variable
- $A^{t,t-1} \in low\_level\_actions$, the action variables
- $O^t_{1:n} \in \{observations\}$, the set of observations variables

$$P(S^{t,t-1}, T^{t,t-1}, A^{t,t-1}, O^t_{1:n}) = P(S^{t-1}).P(T^{t-1}).P(A^{t-1})$$
$$.P(O^t_{1:n}).P(S^t|S^{t-1}, O^t_{1:n}).P(T^t|S^t, T^{t-1}, O^t_{1:n}).P(A^t|T^t, A^{t-1}, O^t_{1:n})$$

Vertical continuity

Vertical continuity in the decision-making process describes when taking a higher-level decision implies a strong conditioning on lower-levels decisions.

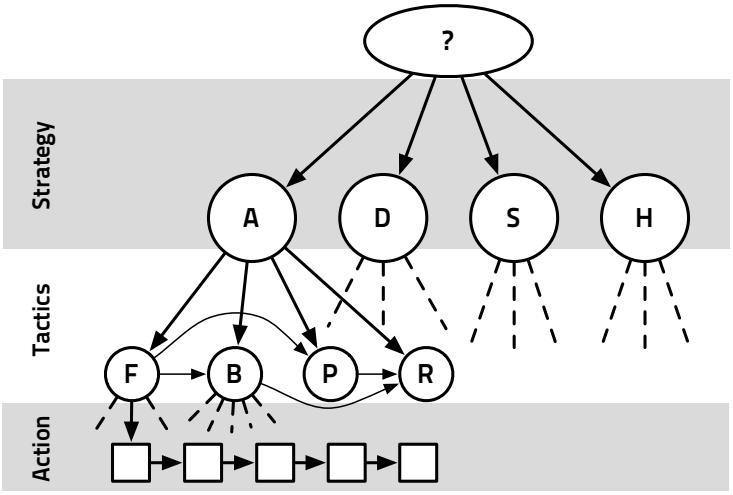


Figure 2.6: Abstract decision hierarchy in a video game

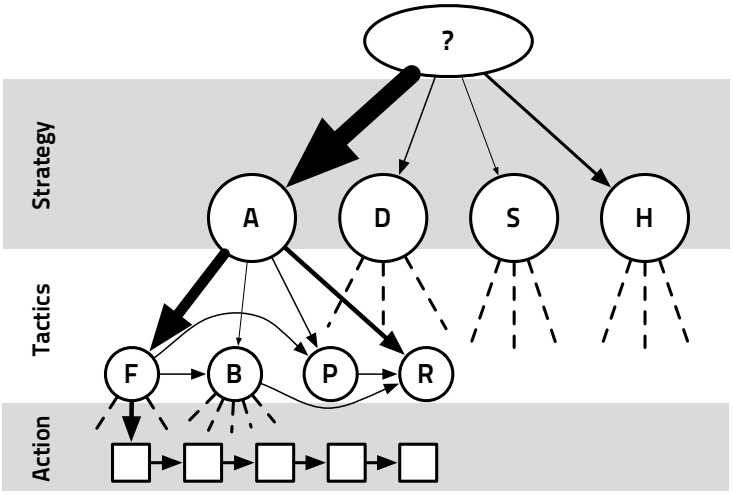


Figure 2.7: Vertical continuity in decision-making in a video game

Horizontal continuity

Partial information

Randomness

Time Constant(s)

Learning Curve

Recap

2.9 Player Characteristics

In all these games, knowledge and learning plays a key role. Humans compensate their lack of (conscious) computational power with pattern matching, abstract thinking and efficient memory structures.

Game	Combinatory	Vertical cont.	Horizontal cont.	Partial Info.	Randomness
Checkers	$b \approxeq 10; n \approxeq 70$	none	none	no	no
Chess	$b \approxeq 35; n \approxeq 80$	none	none	no	no
Go	$b \approxeq 250 - 300; n \approxeq 150 - 200$	none	some	no	no
Limit Poker	$b \approxeq 3^a$; $n/hour \in [20 \dots 240]^b$	some	few	much	much
Time Racing (TrackMania)	$b \approxeq 50 - 1,000^c; n/min \approxeq 60+$	full	much	no	no
Team FPS (Counter-Strike) (Team Fortress 2)	$b \approxeq 100 - 2,000^d$; $n/min \approxeq 100^e$	some	much	some	some
FFPS duel (Quake III)	$b \approxeq 200 - 5,000^d$; $n/min \approxeq 100^e$	some	much	some	(≈)no
MMORPG (WoW, DAoC)	$b \approxeq 50 - 100^f$; $n/min \approxeq 60^g$	much	much	few	moderate
RTS (StarCraft)	$b \approxeq 200^h; n/min = APM \approxeq 300^i$	some	some	much	no

[a] fold,check,raise
[b] number of decisions taken per hour
[c] $\{\ddot{x} \times \theta(\times\phi)\}$ sampling×50Hz
[d] $\{X \times Y \times Z\}$ sampling×50Hz + firing
[e] 60 "continuous move actions"+ 40 (mean) fire actions per sec
[f] in RPGs, the player does not have to aim and positioning plays a lesser role than in FPS
[g] move and use abilities/cast spells
[h] atomic dir/unit × # units + constructions + productions
[i] for pro-gamers, counting group actions as only one action

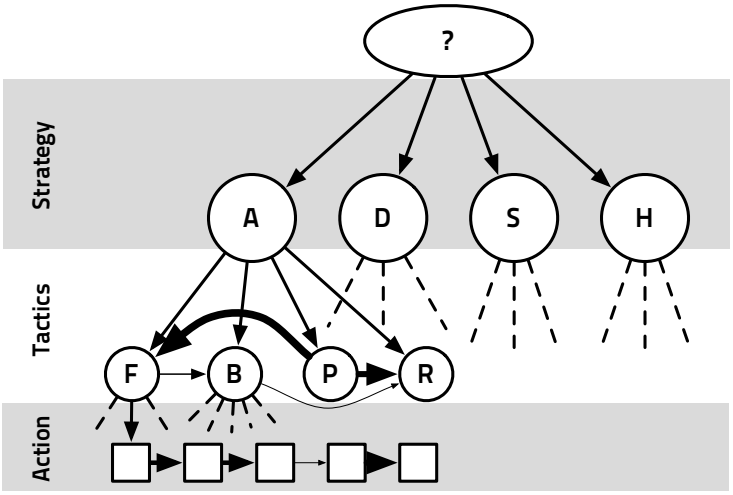


Figure 2.8: Horizontal continuity in decision-making in a video game

Virtuosity

Skill

Deduction

Induction

Decision-Making

Recap

2.10 An interesting problem

Simulated but stochastic

Human players (ally or foes), and sometimes (most of the time) stochasticity in the rules of the game (fog of war, randomness, etc.).

Game	Virtuosity (sensory-motor)	Deduction (analysis)	Induction (abstraction)	Decision-Making (acting)	Knowledge		
					game	map	opponent
Checkers		++			++		+
Chess		++			++		+
Go		++	+		++		+
Limit Poker		+	+	++	++		++
Time Racing (TrackMania)	++				+	++	
Team FPS (Counter-Strike) (Team Fortress 2)							
FFPS duel (Quake III)	++	+		+	+	++	+
MMORPG (WoW, DAoC)	+	+	+	++	+	++	+
RTS (StarCraft)	++	++	++	++	++	+	++

Chapter 3

Bayesian Modeling of Multi-player Games

Question: An efficient and evolutive modeling of a player in a multi-player game. (How do we do...?)

3.1 Transversal problems: summing up problems encountered

Bot AI has to be adaptive, robust, multi-scale.

Why?

- The bot can't cheat (at least it's not fun!)
- The bot can't assume optimal play from the opponent when the problem is so large
- The bot can learn from others games, self past games, self current game
- The bot can't be in the head of your opponent (meta-)

Uncertainty

Uncertainty can arise from game rules (fog-of-war) or from the dimension of the search space (game state space, e.g. Go).

Autonomy

Autonomy is the ability to deal with new states, the challenge of autonomous characters arises from state spaces too big to be fully specified (in scripts / FSM).

3.2 The Bayesian Programming Methodology

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 Diard et al. [2003]. 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 MacKay [2003] and variational Bayes Beal [2003]. 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 Bessière et al. [2008]. For its use in cognitive modeling, see Colas et al. [2010]. For its first use in video games (first person shooter gameplay, Unreal Tournament), see Le Hy et al. [2004].

How?

- Bayesian programming methodology
- When in doubt, toss the distribution to your neighbour
- Exploit gameplay/game rules structure
- Learn and eat data for breakfast
- Meta- can be solved by being (globally) stateless and applying the same model as self on the opponent with her sensory inputs

3.3 Modeling of a Bayesian MMORPG player

We will now present a model of a MMORPG player with the Bayesian programming framework [Synnaeve and Bessière, 2010]. It deals only with a sub-task of a global AI for autonomous NPC. The problem that we try to solve is: how do we choose which skill to use and on which target in a PvE battle? Possible targets are all our allies and foes. Possible skills are all that we know, we will just try and get a distribution over target and skills and pick the most probable combination that is yet possible to achieve (enough energy/mana, no cooldown). For that, we first choose what should be the target given all surrounding variables: is an ally near death that he should be healed, which foe should we focus our attacks on? Once we have the distribution over possible targets, we search the distribution about our skills, pondered by the one on targets. We put extra care in having the same input variables as a human player to keep consistent with our goal of modeling a human. However, some variables can be things that humans subconsciously interpolate from perceptions.

Variables

A very simple set of variables is as follows. We have the n characters as possible targets; each of them has a lot of bound variables. Health/Hit points (HP) are discretized in 10 levels, from the lower to the higher. Distance (D) is discretized in 4 zones around the robot character: contact where it can attack with a contact weapon and then to the further for which we have to move even for shooting the longest distance weapon/spell. Ally (A) is a Boolean variable mentioning if character i is allied with the robot character. Delta hit points (ΔHP) is a 3-valued interpolated value from the previous few seconds of

fight that informs about the ith character getting wounded or healed (or nothing). Imminent death (ID) is also an interpolated value that encodes HP, ΔHP and incoming attacks/attackers. This is a Boolean variable saying if the ith character if going to die anytime soon. This is an example of what we consider that an experienced human player will infer automatically from the screen and notifications. Class (C), simplified over 4 values, gives the type of the ith character: a Tank can take a lot of damages and taunt enemies, a Contact damager can deal huge amounts of damage with contact weapons (rogue, barbarian...), Ranged stands for the class that deals damages from far away (hunters, mages...) and Healers are classes that can heal (in considerable amounts). The Resist variable is the combination of binary variables of resistance to certain types of (magical) damages into one variable. With 3 possible resistances we get (2^3) 8 possible values. For instance "$R_i = FireNat$" means that the ith character resists fire and nature-based damages. Armor (physical damages) could have been included, and the variables could have been separated. The skill variable takes here all the possible skills for the given character, and not only the available ones to cast at the moment to be able to have reusable probability tables (i.e. learnable tables).

- Target: $T \in \{t_1 \dots t_n\}$
- Hit Points: $HP_1 \dots HP_n$ s.t. $HP_i \in [0 \dots 9]$
- Distance: $D_1 \dots D_n$ s.t. $D_i \in \{Contact, Close, Far, VeryFar\}$
- Ally: $A_1 \dots A_n$ s.t. $A_i \in \{true, false\}$
- Derivative Hit Points: $\Delta HP_1 \dots \Delta HP_n$ s.t. $\Delta HP_i \in \{-, 0, +\}$
- Imminent Death: $ID_1 \dots ID_n$ s.t. $ID_i \in \{true, false\}$
- Class: $C_1 \dots C_n$ s.t. $C_i \in \{Tank, Contact, Ranged, Healer\}$
- Resists: $R_1 \dots R_n$ s.t. $R_i \in \{Nothing, Fire, Ice, Nature, FireIce, IceNat, FireNat, All\}$
- Skill: $S \in \{Skill_1 \dots Skill_m\}$

Decomposition

Target selection: we want to compute the probability distribution on the variable Target (T), so we have to consider the joint distribution with all variables on which T is conditionally dependant : T^{t-1} (the previous value of T), and all the variables on each character (except for Resists). The probability of a given target depends on the previous one (it encodes the previous decision and so all previous states). HP_i depends on the facts that the ith character is an ally, on his class, and if he is a target. Such a conditional probability table should be learned, but we can already foresee that a targeted ally with $C = tank$ would have a high probability of having low HP because taking it for target means that we intend to heal him. D_i is more probable to be far if $A_i = false$ and $T = i$ (our kind of druid attack with ranged spells). The probability of the ith character being an ally depends on if we target allies of foes more often. The probability that $\Delta HP_i = -$ is higher for $A_i = false$ and $C_i = healer$ and $T = i$ and also for $A_i = true$ and $C_i = tank$. As for A_i, the probability of ID_i is driven by our soft evidence of targeting characters near death. The probability of C_i is driven by the distribution of foes and allies population, tuned with a soft evidence of which classes our druid human player will target more frequently. Each and every time, if $T \neq i$, the probability of the left variable is given according to the uniform distribution. For the task of computing the distribution on Target, the joint distribution is simplified (by conditional independence of variables) as:

$$P(T, T^{t-1}, HP_{1:n}, D_{1:n}, A_{1:n}, \Delta HP_{1:n}, ID_{1:n}, C_{1:n}) =$$
$$P(T^{t-1}).P(T|T^{t-1}).\prod_{i=1}^{n}[P(HP_i|A_i, C_i, T).P(D_i|A_i, t).P(A_i|T)$$
$$P(\Delta HP_i|A_i, C_i, T).P(C_i|A_i, T).P(ID_i|T)]$$

Skill selection: As previously for targets, we are interested in the conditional probabilities of each character's state variables given other state variables and given T and S. If $T = i$, $S = big\_heal$, $C_i = tank$ and $A_i = true$, the probability that $HP_i = 0$ or 1 (very low) is very high. Some skills have optimal ranges to be used at and so $P(D_i)$ will be affected. $A_i = true$ will have a probability of 1.0 of $S = any\_heal$ as will $A_i = false$ have a probability of 1.0 is $S = any\_damage$. The probability of $\Delta HP_i = -$ will top when $S = heal$ for an ally. The one of $R_i = nature$ for $S = nature\_damage$ will be very low. The probability of ID_i will be high for $T = i$ and $S = big\_heal$ or $S = big\_damage$ (depending on whether i is an ally or not). For the task of computing the distribution on Skill we use:

$$P(S, T, HP_{1:n}, D_{1:n}, A_{1:n}, \Delta HP_{1:n}, ID_{1:n}, C_{1:n}, R_{1:n}) =$$
$$P(S).P(T).\prod_{i=1}^{n}[P(HP_i|A_i, C_i, S, T).P(D_i|A_i, S, T).P(A_i|S, T)$$
$$P(\Delta HP_i|A_i, S, T).P(R_i|C_i, S, T).P(C_i|A_i, S, T)]$$

Parameters

- $P(T^{t-1})$ Unknown and unspecified (uniform).
- $P(T|T^{t-1})$ Table, specified with a "prior" to prevent switching targets too often or simply learned. Uniform if there is no previous target.
- $P(S)$ Unknown and so unspecified, it could be a prior.
- $P(Left\_Value|Right\_Value)$ All others are *learned tables*.

Identification

If there were only perceived variables, learning the right conditional probability tables would just be counting and averaging. However, some variables encode combinations of perceptions and passed states. We could learn such parameters through the EM algorithm but we propose something simpler for the moment as our "not directly observed variables" are not complex to compute, we compute them from perceptions as the same time as we learn. In the following Results part, we did not apply learning but instead manually specified the probability tables.

Questions

In any case, we ask our model:

$$P(S, T|hp_{1:n}, d_{1:n}, a_{1:n}, \Delta hp_{1:n}, id_{1:n}, c_{1:n}, r_{1:n})$$

Which means that we want to know the distribution on S and T knowing all the state variables. We then choose to do the highest scoring combination of $S \wedge T$ that is available (skills may have cooldowns or cost more mana/energy that we have available).

As (Bayes rule) $P(S, T) = P(S|T).P(T)$, to decompose this question, we can ask:

$$P(T|hp_{1:n}, d_{1:n}, a_{1:n}, \Delta hp_{1:n}, id_{1:n}, c_{1:n})$$

Which means that we want to know the distribution on T knowing all the relevant state variables, followed by (with the newly computed distribution on T):

$$P(S|T, hp_{1:n}, d_{1:n}, a_{1:n}, \Delta hp_{1:n}, id_{1:n}, c_{1:n}, r_{1:n})$$

in which we use this distribution on T to compute the distribution on S with:

$$P(S = skill_1|\dots) = \sum_T P(S = skill_1|T, \dots).P(T)$$

We here choose to sum over all possible values of T. Note that we did not ask:
$P(S|T = most\_probable, \dots)$ but computed instead

$$\sum_T P(S|T, hp_{1:n}, d_{1:n}, a_{1:n}, \Delta hp_{1:n}, id_{1:n}, c_{1:n}, r_{1:n})$$

This computation has a high complexity (particularly when the sum has a lot of term, i.e. with a lot of targets), so we could choose not to do the sum and use and instantiate "most probable values", for instance of Target, but there we would make a choice earlier and so lose information. There are possibly good combinations of S and T for a value of T that is not the most probable one. This downside may be so hard that we may want to reduce the complexity of computation by simplifying our model or its computation to be able to sum. We propose a solution in the discussion.

Example

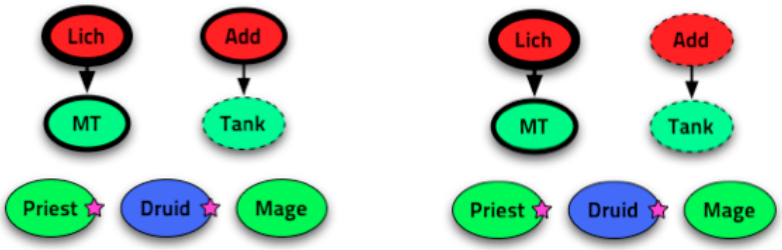


Figure 3.1: Example setup A (left) and B (right), 2 foes, 2 "tanks", players with stars can heal allies, players with dotted lines will soon die ($ID = true$).

This model has been applied to a simulated situation with 2 foes and 4 allies while our robot took the part of a "druid", a versatile class that can cast spells to do direct damages, damages over time, buff (enhancements), debuff, crowd-control, heal and heal over time. We display a schema of this situation in Fig. 3.1 The arrows indicate foes attacks on allies. The larger the ring is, the more health points the characters have. MT stands for "main tank", Add for "additional foe". We worked with the skills corresponding to a Druid. HOT stands for heal over time, DOT for damage over time, "abol" for abolition and "regen" for regeneration, a "buff" is an enhancement and a "dd" is a direct damage. "Root" is a spell which disables the target to move for a short period of time, useful to flee or to put some distance between the enemy and the druid to cast attack spells. "Small" spells are usually faster to cast than "big" spells.

$$Skills \in \{small\_heal, big\_heal, HOT, poison\_abol, malediction\_abol,$$
$$buff\_armor, regen\_mana, small\_dd, big\_dd, DOT, debuff\_armor, root\}$$

We did not do the "Identification" part, which consists in learning the probability tables from observations. To keep things simple and because we wanted to analyze a little the model, we worked with

manually defined probability tables. So we introduced "soft evidences", indeed parameters that will modify the conditional probability tables, which we will change to watch their effects. For instance the "soft evidence that a selected target is foe" and the "soft evidence that a selected target will soon die ($ID = true$)" that will consequently modify the probability tables of $P(A_i)$ and $P(ID_i)$ respectively. We set the probability to target the same target as before to 0.4 and the previous target to "Lich" so that the prior probability for all other 6 targets is 0.1 (4 times more chances to target the Lich than any other character). We set the soft evidence $P(A_i = false|T = i)$ to 0.6. This means that our robotic druid is mainly a damage dealer and not a healer. For the "target selection" model, we can see on Fig. 3.2 (left) that the evolution from selecting the main foe "Lich" to selecting the ally "Tank" is driven by the increase of "soft evidence that a selected target will soon die" and our robot eventually moves on targeting his(its) "Tank" ally (to heal him). We can see on Fig. 3.2 (right) that, at some point, the robotic Druid prefers to kill the dying add to save his ally Tank instead of healing him. Note that there is no variable showing the relation between "Add" and "Tank" (the first is attacking the second, who is taking damages from the first), but this is under consideration for a future, more complete, model.

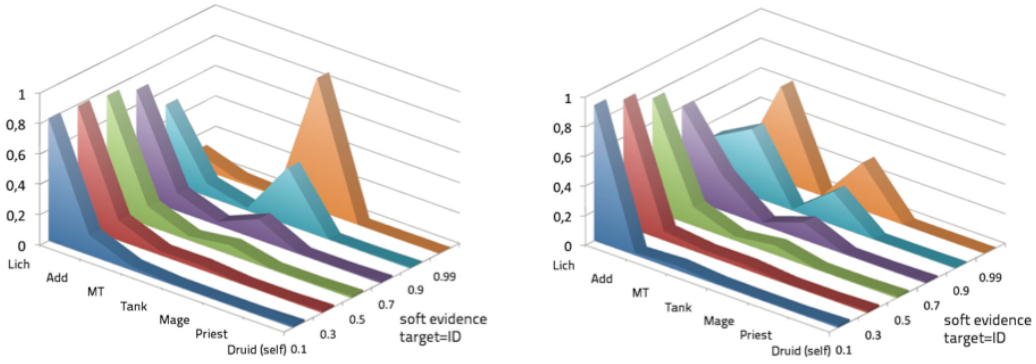


Figure 3.2: Left: probabilities of targets depending on the soft evidence that a target is dying with setup A. Right: same, with setup B.

For the "skill selection" model, we can see on Fig. 3.3 the influence of ID_i on Skill which is coherent with the Target distribution: either, in setup A (left), we evolve with the increase of $P(ID_i = true|Target = i)$ to choose to heal our ally or, in setup B (right), to deal direct damage (and hopefully, kill) the foe attacking him. As you can see here, when we have the highest probability to attack the main enemy ("Lich", when $P(ID_i = true|Target = i)$ is low), who is a $C = tank$, we get a high probability for the Skill $debuff\_armor$. We only cast this skill if the debuff is not already present, so perhaps that we will cast $small\_dd$ instead. To conclude this example, Fig. 3.4 shows the distribution on $P(T, S|all\_status\_variables)$ with setup A and the probability to target the previous target (set to "Lich" here) only $\approxeq 2$ times greater than any other character (so that we focus less on the same character), soft evidences $P(ID_i = true|Target = i) = 0.9$ and $P(A_i = false|Target = i) = 0.6$. In a greedy way, if the first couple (T, S) is already done or not available, we take the second.

Discussion

This model has to be applied in a real MMORPG, out of its simulated sandbox, to reveal all its shortcomings and be updated. We can already think of some future difficulties, for instance there is a possibility for many games that the Skill variable will be very big and that it will imply a too high computational cost. For that concern, we propose to clusterize the skills in global skills (GS) (as it can be seen in the description of the example). This approach to break down the complexity of computation is general and can be used with other variables. The skill variable S can then be the subset of skills corresponding to the clustering of GS, for instance we could have:

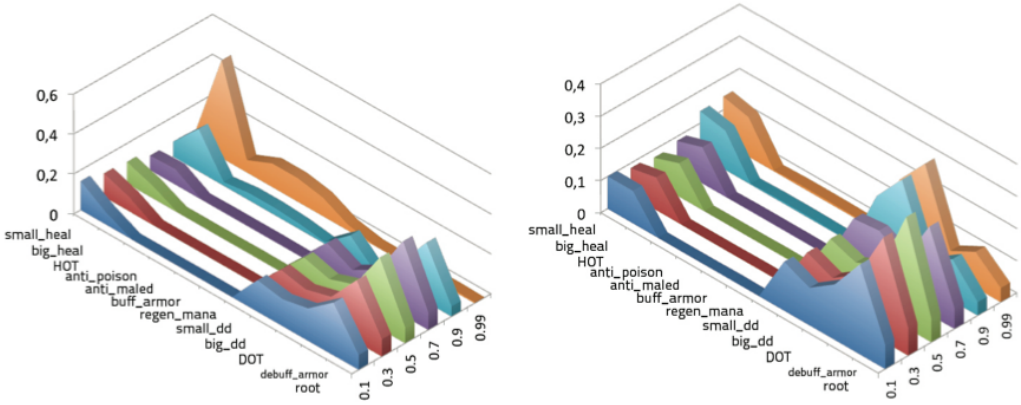


Figure 3.3: Left: Probabilities of skills depending on the soft evidence that a target is dying with setup A. Right: same, with setup B.

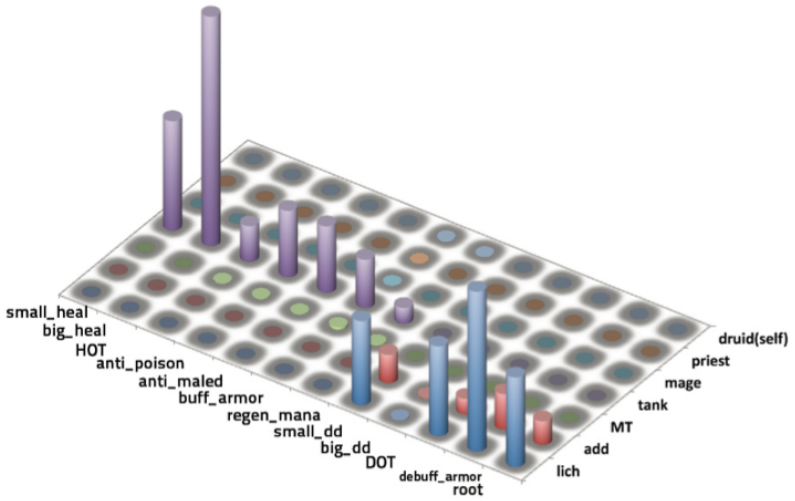


Figure 3.4: Log-probabilities of Target and Skill with setup A, and $P(ID|Target) = 0.9, P(A|Target) = 0.6$

$$GS \in \{SkillHeal, SkillBuff, SkillAttack, SkillDebuff\}$$

$$S = SkillHeal \in \{skill_1 \dots skill_j\}$$

$$S = SkillBuff \in \{skill_{j+1} \dots skill_k\}$$

$$S = SkillAttack \in \{skill_{k+1} \dots skill_l\}$$

$$S = SkillDebuff \in \{skill_{l+1} \dots skill_m\}$$

Global skills joint distribution: as for the "Skill joint distribution" without Resists. It will take advantage of splitting between allies and foes.

$$P(GS, T, HP_{1:n}, D_{1:n}, A_{1:n}, \Delta HP_{1:n}, ID_{1:n}, C_{1:n}) =$$
$$P(GS).P(T).\prod_{i=1}^{n}[P(HP_i|A_i, C_i, GS, T).P(D_i|A_i, GS, T).P(A_i|GS, T)$$
$$.P(\Delta HP_i|A_i, GS, T).P(ID_i|GS, T).P(C_i|A_i, GS, T)]$$

Specialized skills joint distribution:

$$P(S, GS, T, HP_{1:n}, D_{1:n}, A_{1:n}, \Delta HP_{1:n}, ID_{1:n}, C_{1:n}, R_{1:n}) =$$
$$P(S|GS).P(T).\prod_{i=1}^{n}[P(HP_i|A_i, C_i, S, T).P(D_i|A_i, S, T).P(A_i|S, T)$$
$$.P(\Delta HP_i|A_i, S, T).P(R_i|C_i, S, T).P(ID_i|S, T).P(C_i|A_i, S, T)]$$

for the corresponding S. So that we can ask the question:

$$P(S|GS, T, hp_{1:n}, d_{1:n}, a_{1:n}, \Delta hp_{1:n}, id_{1:n}, c_{1:n}, r_{1:n})$$

that will trigger $P(GS|T, \dots)$, itself triggering $P(T|all\_state\_variables)$. Choosing to do with or without the intermediate GS computation, regrouping abilities by types, is mainly a question of computational time.

The choice of the skill or ability to use and the target on which to use it puts hard constraints on every others decisions the autonomous agent has to take to perform its ability action. Thus, such a model shows that:

- cooperative behavior is not too hard to incorporate in a decision (instead of being hard-coded),
- it can be learned, either from observations of a human player or by reinforcement (exploration),
- it is computationally tractable (for use in all games), the inference is just a series of "probabilistic *if*s",

Moreover, using this model on another agent than the once controlled by the AI can give a prediction on what it will do, resulting in human-like, adaptive, playing style.

We did not kept at the research track of Bayesian modeling MMORPG games due to the difficulty to work on these types of games: the studios have too much to lose to farmer bots to accept any automated access to the game. Also, there are no sharing format of data (like replays) and the invariants of the game situations are fewer than in RTS games. Finally, RTS games have international AI competitions which were a good motivation to compare our approach with other game AI researchers.

Chapter 4

RTS AI: *StarCraft: Broodwar*

4.1 How does the game works: gameplay

Really, the best is to play it... But I'll explain it.

In combinatorial game theory terms, competitive StarCraft is a zero sum, partial-information, deterministic strategy game.

4.2 Problems, resolutions

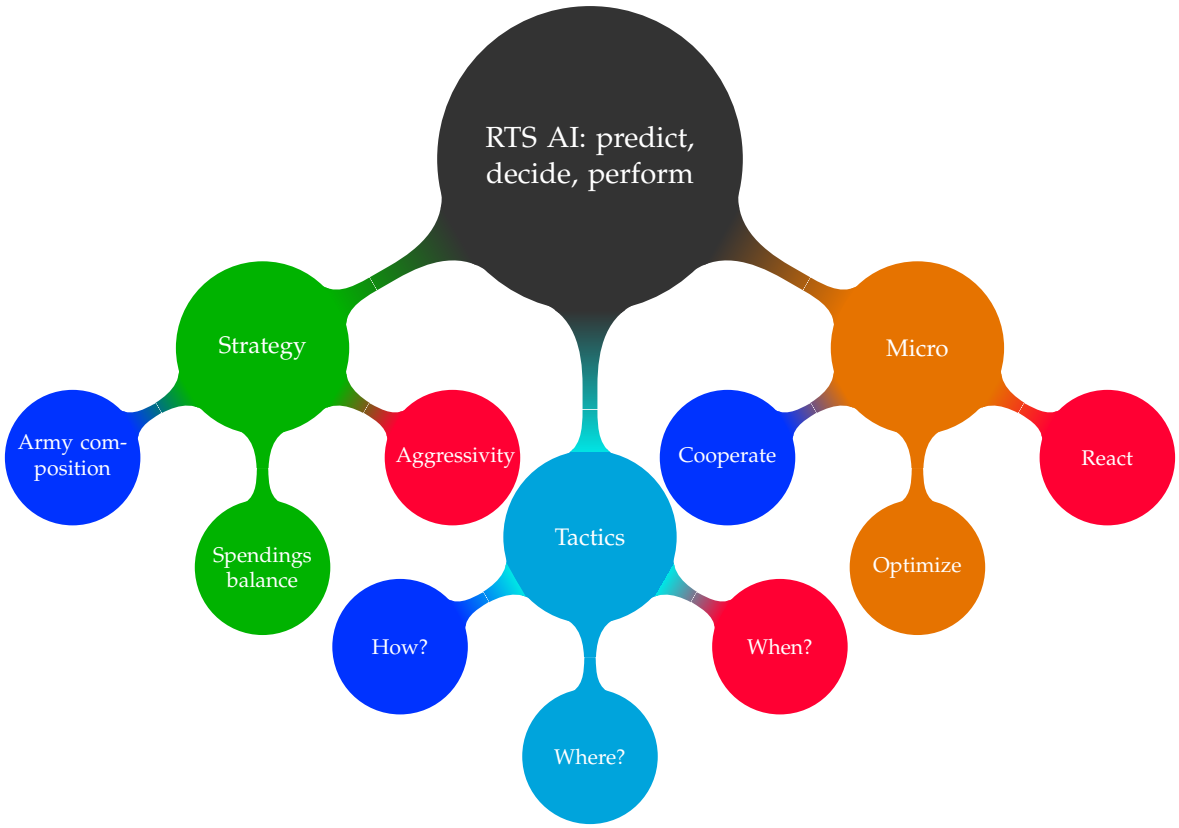


Figure 4.1: A mind-map of RTS AI XXX TODO

4.3 Task decomposition and linking

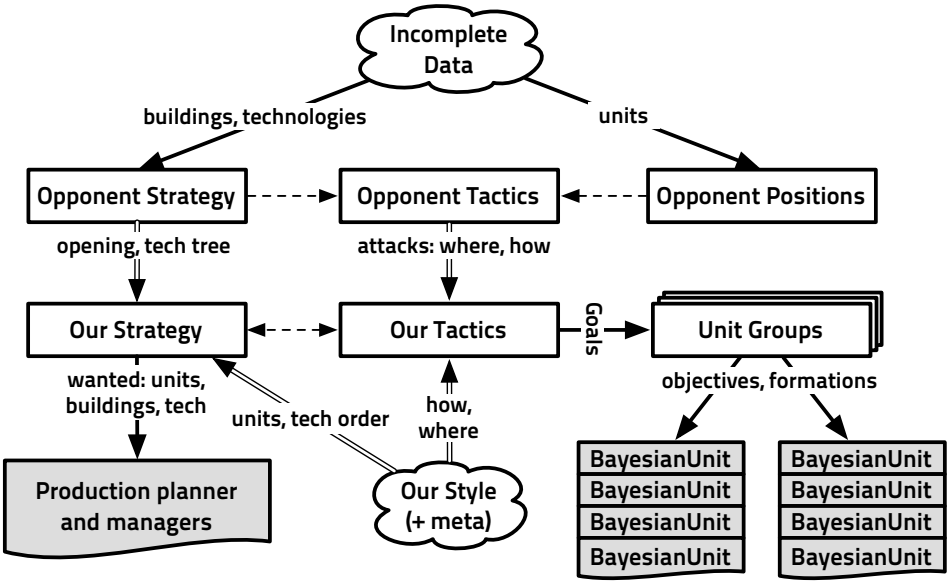


Figure 4.2: Information-centric view of the architecture of the major components of the bot. Arrows are labeled with the information or orders they convey: dotted arrows are convey constraints, double lined arrows convey distributions, plain and simple arrows convey direct information or orders. The gray parts perform game actions (as the physical actions of the player on the keyboard and mouse).

In Fig. 4.3, we present the flow of informations between the different inference and decision-making parts of the bot architecture. One can also view this problem as having a good model of one's strategy, one's opponent strategy, and taking decisions. The software architecture that we propose is to have services building and maintaining the model of the enemy as well as our state, and decision-making modules using all this information to give orders to actuators.

- Problem: build a real-scale software piece which is maintainable
- State of the art: shared memories, shared states
- Our take: we transmit distributions, states stay in modules
- Results: XXX (atm too much state), also competitions results

XXX Real-time strategy (RTS) gameplay consist in producing and managing group of units with attacks and movements specificities in order to defeat an enemy. Most often, it is required to gather resources and build up an economic and military power while expanding a technology tree. Parts of the map not in the sight range of the player's units are under *fog of war*, so the player only has partial information about the enemy buildings and army. The way by which we expand the tech tree, the specific units composing the army, and the general stance (aggressive or defensive) form what we call *strategy*. At the lower level, the actions performed by the player (human or not) to optimize the effectiveness of its units is called *micro-management*. In between lies *tactics*: where to attack, and how. A good human player takes much data in consideration when choosing: are there flaws in the defense? Which spot is more worthy to attack? How much am I vulnerable for attacking here? Is the terrain (height, chokes) to my advantage? etc.

Chapter 5

Micro-management

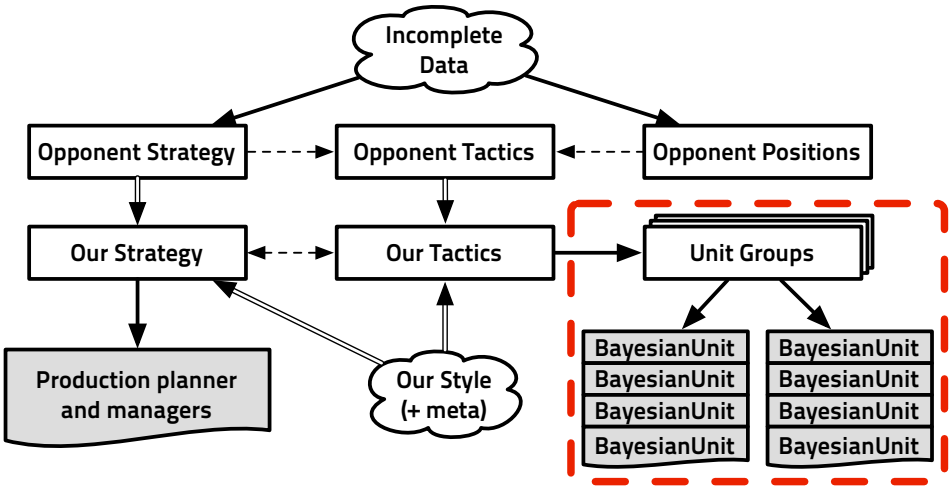


Figure 5.1: Information-centric view of the architecture of the bot, the part concerning this chapter is in the dotted rectangle

5.1 Units Management

Problem: (optimal) control of units in a (real-continuous-time) huge actions space Complexity: P-space

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 Lebeltel et al. [2004] 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.

5.2 Related Works

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) [Rabin, 2002], genetic algorithms (GA) [Ponsen and Spronck, 2004, Preuss et al., 2010], reinforcement learning (RL) [Marthi et al., 2005, Madeira et al., 2006], case-based reasoning (CBR) [Aha et al., 2005, Sharma et al., 2007], continuous action models [Molineaux et al., 2008], reactive planning [Weber et al., 2010b], upper confidence bounds tree (UCT) [Balla and Fern, 2009], potential fields [Hagelbäck and Johansson, 2009], influence maps[Preuss et al., 2010], and cognitive human-inspired models [Wintermute et al., 2007].

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 [Cutumisu and Szafron, 2009]. Hierarchical FSM (HFSM) solve some of this problems (state sharing) and evolved into behavior trees (BT, hybrids HFSM) [Isla, 2005] and behavior multi-queues (resumable, better for animation) [Cutumisu and Szafron, 2009] 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 [Sutton and Barto, 1998] to RTS research are concurrent hierarchical (units Q-functions are combined higher up) RL [Marthi et al., 2005] to efficiently control units in a multi-effector system fashion, and large-scale strategy games [Madeira et al., 2006]. 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 [Marthi et al., 2005]. 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 [Ponsen and Spronck, 2004].

Case-based reasoning (CBR) allows for learning against dynamic opponents [Aha et al., 2005] and has been applied successfully to strategic and tactical planning down to execution through behavior reasoning rules [Ontañón et al., 2007]. 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 [Molineaux et al., 2008], an integrated RL/CBR algorithm using continuous models, and reactive planning [Weber et al., 2010b], 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 [Chung et al., 2005] and upper Upper confidence bounds tree (UCT) planning (coming from Go AI) [Balla and Fern, 2009] samples through the (rigorously intractable) plans space by incrementally building the actions tree through Monte-Carlo sampling. UCT for tactical assault planning [Balla and Fern, 2009] 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 [Hagelbäck and Johansson, 2009], 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 [Preuss et al., 2010]. 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 [Wintermute et al., 2007], 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).

5.3 A Bayesian Model for Units Control

[Synnaeve and Bessière, 2011a]

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.

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 [Balla and Fern, 2009]. 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.

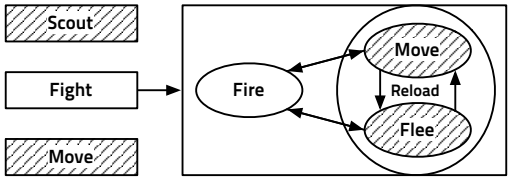


Figure 5.2: Bayesian unit modal FSM, detail on the fight mode. Stripped modes are Bayesian.

We propose to model units as sensory-motor robots described within the Bayesian robot programming framework [Lebeltel et al., 2004]. 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:

Variables

- $Dir_{i \in [\![0 \dots n]\!]} \in \{True, False\}$: 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.

- $Obj_{i\in[\![0\dots n]\!]} \in \{True, False\}$: 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).

- $Dmg_{i\in[\![0\dots n]\!]} \in [DamageValues]$ for instance, with $ubhp$ standing as unit base hit points, $Dmg_i \in \left\{0, [\![0 \dots \frac{ubhp}{2}]\!],]\!]\frac{ubhp}{2} \dots ubhp[\!], [\![ubhp \cdots + \inf[\!]\right\}$. This will act as subjective potential fields [Hagelbäck and Johansson, 2009] 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.

- $A_{i\in[\![0\dots n]\!]} \in \{None, Small, Big\}$: 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).

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

- $Occ_{i\in[\![0\dots n]\!]} \in \{None, Building, StaticTerrain\}$ (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.

Decomposition

The joint distribution (JD) over these variables is a specific kind of fusion called inverse programming [Le Hy et al., 2004]. 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.

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.

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).
- $Dir^{t-1}_{i\in[\![0...n]\!]} \in \{True, False\}$: 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 $JD \times P(Dir).\Pi_{i=1}^{n}P(Dir_i|Dir)$.

Identification

Parameters and probability tables can be learned through reinforcement learning [Sutton and Barto, 1998, Asmuth et al., 2009] 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.

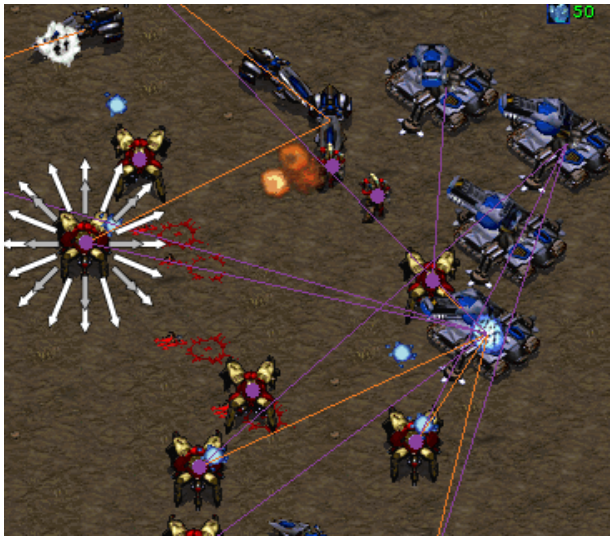


Figure 5.3: Screen capture of a fight in which our bot controls the bottom-left units in StarCraft. The 24 possible directions are represented for a unit with white and grey arrows.

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.

5.4 Results on StarCraft

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.

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. Their objectives are set by military goal-wise atomic units group, themselves spawned to achieve tactical goals (see Fig. 5). Units groups tune their Bayesian units modes (scout,

fight, move) and give 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.

Experiments

Our implementation[1] (BSD licensed) uses BWAPI[2] 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%)

Figure 5.4: 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.

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):

[1] BROODWARBOTQ, code and releases: `http://github.com/SnippyHolloW/BroodwarBotQ`
[2] BWAPI: `http://code.google.com/p/bwapi/`

- 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*.

5.5 Extensions

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 reactive planning [Ontañón et al., 2007, Weber et al., 2010b], 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 [Rabin, 2002].

Future work could consist in using reinforcement learning [Sutton and Barto, 1998] or evolutionary algorithms [Smith et al., 2010] to learn the probability tables. It should enhance the performance of our Bayesian units in specific setups. It implies making up challenging scenarii and dealing with huge sampling spaces [Asmuth et al., 2009]. Also, we could use multi-modality [Colas et al., 2010] and inverse programming [Le Hy et al., 2004] 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.

Chapter 6

Tactics

In their study on human like characteristics in RTS games, Hagelbäck and Johansson Hagelbäck and Johansson [2010] found out that "tactics was one of the most successful indicators of whether the player was human or not".

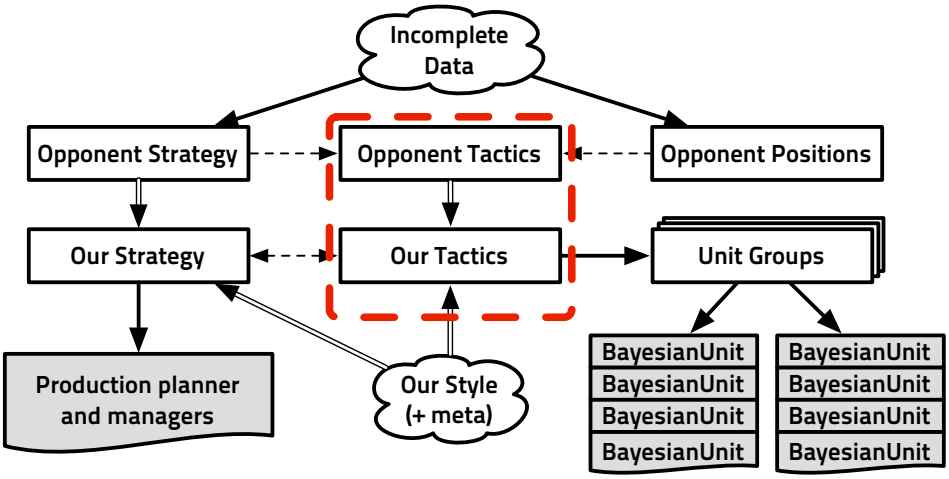


Figure 6.1: Information-centric view of the architecture of the bot, the part concerning this chapter is in the dotted rectangle

- Problem: choose which tactical actions/goals to persue, peform the action
- Complexity: incompleteness/uncertainty problem, lot of low level information to handle, w.r.t. full and higher level information, simple.
- State of the art: [Wintermute et al., 2007, Weber et al., 2010a, Balla and Fern, 2009, Cadena and Garrido, 2011]
- Our take: low level heuristics that we learn to adapt to
- Results: XXX
- Conclusion and perspectives: still enabled for meta-game, and even in-game, adaptation. Could learn the action sequences of tactics from replays (≈HMM).

6.1 What are Tactics?

In this part, we focus on tactics, in between strategy (high-level) and micro-management (lower-level), as seen in Fig. 6.2. We propose a model which can either predict enemy attacks or give us a distribution on where and how to attack the opponent. Information from the higher-level strategy constrains what types of attacks are possible. As shown in Fig. 6.2, information from units positions (or possibly an enemy units particle filter as in **?**) constrains where the armies can possibly be in the future. In the context of our StarCraft bot, once we have a decision: we generate a goal (attack order) passed to units groups (see Fig.6). A Bayesian model for micro-management Synnaeve and Bessière [2011a], in which units are attracted or repulsed by dynamic (goal, units, damages) and static (terrain) influence maps, actually moves the units in StarCraft. Other previous works on strategy prediction Synnaeve and Bessière [2011], Synnaeve and Bessière [2011b] allows us to infer the enemy tech tree and strategies from incomplete information (due to the fog of war).

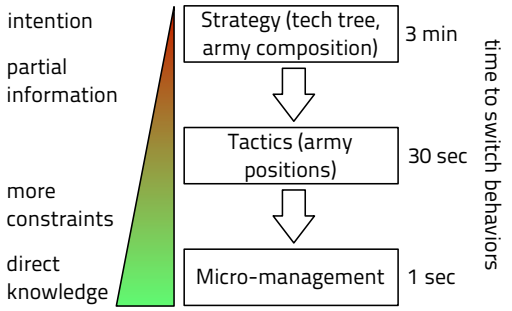


Figure 6.2: Gameplay levels of abstraction for RTS games, compared with their level of direct (and complete) information and orders of magnitudes of time to chance their policies.

Units have different abilities, which leads to different possible tactics. Each faction has invisible (temporarily or permanently) units, flying transport units, flying attack units and ground units. Some units can only attack ground or air units, some others have splash damage attacks, immobilizing or illusion abilities. Fast and mobile units are not cost-effective in head-to-head fights against slower bulky units. We used the gamers' vocabulary to qualify different types of tactics: *ground* attacks (raids or pushes) are the most normal kind of attacks, carried by basic units which cannot fly. Then comes *air* attacks (air raids), which use flying units mobility to quickly deal damage to undefended spots. *Invisible* attacks exploit the weaknesses (being them positional or technological) in detectors of the enemy to deal damage without retaliation. Finally, *drops* are attacks using ground units transported by air, combining flying units mobility with cost-effectiveness of ground units, at the expense of vulnerability during transit.

6.2 Related Works

Aha et al. Aha et al. [2005] used case-based reasoning (CBR) to perform dynamic tactical plan retrieval (matching) extracted from domain knowledge in Wargus. Ontañó et al. **?** based their real-time case-based planning (CBP) system on a plan dependency graph which is learned from human demonstration in Wargus. A case based behavior generator spawn missing goals which are missing from the current state and plan according to the recognized state. In **??**, they used a knowledge-based approach to perform situation assessment to use the right plan, performing runtime adaptation by monitoring its performance. Sharma et al. Sharma et al. [2007] combined CBR and reinforcement learning to enable reuse of tactical plan components. Cadena and Garrido Cadena and Garrido [2011] used fuzzy CBR (fuzzy case matching) for strategic and tactical planning. Chung et al. Chung et al. [2005] adapted

Monte-Carlo tree search (MCTS) to planning in RTS games and applied it to a capture-the-flag mod of Open RTS. Balla and Fern **?** applied upper confidence bounds on trees (UCT: a MCTS algorithm) to tactical assault planning in Wargus.

In Starcraft, Weber et al. Weber et al. [2010a,b] produced tactical goals through reactive planning and goal-driven autonomy, finding the more relevant goal(s) to follow in unforeseen situations. Kabanza et al. **?** performs plan and intent recognition to find tactical opportunities. On spatial and temporal reasoning, Forbus et al. Forbus et al. [2002] presented a tactical qualitative description of terrain for wargames through geometric and pathfinding analysis. Perkins Perkins [2010] automatically extracted choke points and regions of StarCraft maps from a pruned Voronoi diagram, which we used for our regions representations. Wintermute et al. Wintermute et al. [2007] used a cognitive approach mimicking human attention for tactics and units control. Ponsen et al. **?** developed an evolutionary state-based tactics generator for Wargus. Finally, Avery et al. **?** and Smith et al. Smith et al. [2010] co-evolved influence map trees for spatial (tactical) reasoning in RTS games.

Our approach (and bot architecture, depicted in Fig. 6) can be seen as goal-driven autonomy Weber et al. [2010a] dealing with multi-level reasoning by passing distributions (without any assumption about how they were obtained) on the module input. Using distributions as messages between specialized modules makes dealing with uncertainty first class, this way a given model do not care if the uncertainty comes from incompleteness in the data, a complex and biased heuristic, or another probabilistic model. We then take a decision by sampling or taking the most probable value in the output distribution. Another particularity of our model is that it allows for prediction of the enemy tactics using the same model with different inputs. Finally, our approach is not exclusive to most of the techniques presented above, and it could be interesting to combine it with UCT **?** and more complex/precise tactics generated through planning.

6.3 A Bayesian Tactical Model

Dataset

We downloaded more than 25,000 replays to keep 22,947 uncorrupted, 1v1 replays of very high level StarCraft games (pro-gamers leagues and international tournaments) from specialized websites[1][2][3], we then ran them using BWAPI[4] and dumped units positions, pathfinding and regions, resources, orders, vision events, for attacks (we trigger an attack tracking heuristic when one unit dies and there are at least two military units around): types, positions, outcomes. Basically, every BWAPI event was recorded, the dataset and its source code are freely available[5].

We used two kinds of regions: BroodWar Terrain Analyser (BWTA) regions and choke-dependent (choke-centered) regions. BWTA regions are obtained from a pruned Voronoi diagram on walkable terrain Perkins [2010] and gives regions for which chokes are the boundaries. As battles often happens at chokes, choke-dependent regions are created by doing an additional (distance limited) Voronoi tesselation spawned at chokes, its regions set is $(regions \setminus chokes) \cup chokes$. Results for choke-dependent regions are not fully detailed.

Tactical Model

The idea is to have (most probably biased) lower-level heuristics from units observations which produce information exploitable at the tactical level, and take some advantage of strategic inference too. The advantages are that 1) learning will de-skew the model output from biased heuristic inputs 2) the

[1] `http://www.teamliquid.net`
[2] `http://www.gosugamers.net`
[3] `http://www.iccup.com`
[4] `http://code.google.com/p/bwapi/`
[5] `http://snippyhollow.github.com/bwrepdump/`

model is agnostic to where input variables' values come from 3) the updating process is the same for supervised learning and for reinforcement learning.

We note $s^{a|d}_{unit\ type}(r)$ for the balanced score of units from attacker or defender ($^{a|d}$) of a given type in region r. The balanced score of units is just the sum of units multiplied by each unit score ($= minerals\_value + \frac{4}{3}gas\_value + 50supply\_value$). The heuristics we used in our benchmarks (which we could change) are:

$$economical\_score^d(r) = \frac{s^d_{workers}(r)}{\sum_r s^d_{workers}(r)}$$

$$tactical\_score^d(r) = \sum_{i \in regions} s^d_{army}(i) \times dist(i,r)^{-1.5}$$

We used $^{-1.5}$ such that the tactical value of a region in between two halves of an army, each at distance 2, would be higher than the tactical value of a region at distance 4 of the full (same) army. For flying units, *dist* is the Euclidean distance, while for ground units it takes pathfinding into account.

$$ground\_defense^d(r) = \frac{s^d_{can\_attack\_ground}(r)}{s^a_{ground\_units}(r)}$$

$$air\_defense^d(r) = \frac{s^d_{can\_attack\_air}(r)}{s^a_{air\_units}(r)}$$

$$invis\_defense^d(r) = number^d_{detectors}$$

We preferred to discretize continuous values to enable quick complete computations. An other strategy would keep more values and use Monte Carlo sampling for computation. We think that discretization is not a concern because 1) heuristics are simple and biased already 2) we often reason about imperfect information and this uncertainty tops discretization fittings.

Variables

With n regions, we have:

- $A_{1:n} \in \{true, false\}$, A_i: attack in region i or not?
- $E_{1:n} \in \{no, low, high\}$, E_i is the discretized economical value of the region i for the defender. We choose 3 values: *no* workers in the regions, *low*: a small amount of workers (less than half the total) and *high*: more than half the total of workers in this region i.
- $T_{1:n} \in dicrete\ levels$, T_i is the tactical value of the region i for the defender, see above for an explanation of the heuristic. Basically, T is proportional to the proximity to the defender's army. In benchmarks, discretization steps are $0, 0.05, 0.1, 0.2, 0.4, 0.8$ (log_2 scale).
- $TA_{1:n} \in dicrete\ levels$, TA_i is the tactical value of the region i for the attacker (see above).
- $B_{1:n} \in \{true, false\}$, B_i tells if the region belongs (or not) to the defender. $\mathrm{P}(B_i = true) = 1$ if the defender has a base in region i and $\mathrm{P}(B_i = false) = 1$ if the attacker has one. Influence zones of the defender can be measured (with uncertainty) by $\mathrm{P}(B_i = true) \geq 0.5$ and vice versa.
- $H_{1:n} \in \{ground, air, invisible, drop\}$, H_i: in predictive mode: how we will be attacked, in decision-making: how to attack, in region i.
- $GD_{1:n} \in \{no, low, med, high\}$: ground defense (relative to the attacker power) in region i, result from a heuristic. *no* defense if the defender's army is $\geq 1/10th$ of the attacker's, *low* defense above that and under half the attacker's army, *medium* defense above that and under comparable sizes, *high* if the defender's army is bigger than the attacker.

- $AD_{1:n} \in \{no, low, med, high\}$: same for air defense.
- $ID_{1:n} \in \{no\ detector, one\ detector, several\}$: invisible defense, equating to numbers of detectors.
- $TT \in [\emptyset, building_1, building_2, building_1 \wedge building_2, techtrees, \dots]$: all the possible technological trees for the given race. For instance $\{pylon, gate\}$ and $\{pylon, gate, core\}$ are two different Tech Trees.
- $P \in \{ground, ground \wedge air, ground \wedge invis, ground \wedge air \wedge invis, ground \wedge drop, ground \wedge air \wedge drop, ground \wedge invis \wedge drop, ground \wedge air \wedge invis \wedge drop\}$: possible types of attacks, directly mapped from TT information. In prediction, with this variable, we make use of what we can infer on the opponent's strategy Synnaeve and Bessière [2011b], Synnaeve and Bessière [2011], in decision-making, we know our own possibilities (we know our tech tree as well as the units we own).

Finally, for some variables, we take uncertainty into account with "soft evidences": for instance for a region in which no player has a base, we have a soft evidence that it belongs more probably to the player established closer. In this case, for a given region, we introduce the soft evidence variable(s) B' and the coherence variable λ_B and impose $\mathrm{P}(\lambda_B = 1|B, B')\ iff\ B = B'$, while $\mathrm{P}(\lambda_B|B, B').\mathrm{P}(B')$ is a new factor in the joint distribution. This allows to sum over $\mathrm{P}(B')$ distribution (soft evidence).

Decomposition

The joint distribution of our model contains soft evidence variables for all input family variables (E, T, TA, B, GD, AD, I to be as general as possible, *i.e.* to be able to cope with all possible uncertainty (from incomplete information) that may come up in a game. To avoid being too verbose, we explain the decomposition only with the soft evidence for the family of variables B, the principle holds for all other soft evidences. For the n considered regions, we have:

$$\begin{aligned} & \mathrm{P}(A_{1:n}, E_{1:n}, T_{1:n}, TA_{1:n}, B_{1:n}, B'_{1:n}, \lambda_{B,1:n}, \\ & H_{1:n}, GD_{1:n}, AD_{1:n}, ID_{1:n}, P, TT) \\ = & \prod_{i=1}^{n} [\mathrm{P}(A_i).\mathrm{P}(E_i, T_i, TA_i, B_i|A_i). \qquad (1) \\ & \mathrm{P}(\lambda_{B,i}|B_{1:n}, B'_{1:n}).\mathrm{P}(B'_{1:n}). \\ & \mathrm{P}(AD_i, GD_i, ID_i|H_i).\mathrm{P}(H_i|P)].\mathrm{P}(P|TT).\mathrm{P}(TT) \end{aligned}$$

Forms and Learning

We will explain the forms for a given/fixed i region number:

- $\mathrm{P}(A)$ is the prior on the fact that the player attacks in this region, in our evaluation we set it to $n_{battles}/(n_{battles} + n_{not\ battles})$. In a given match it should be initialized to uniform and progressively learn the preferred attack regions of the opponent for predictions, learn the regions in which our attacks fail or succeed for decision-making.
- $\mathrm{P}(E, T, TA, B|A)$ is a covariance table of the economical, tactical (both for the defender and the attacker), belonging scores where an attacks happen. We just use Laplace succession law ("add one" smoothing) **?** and count the co-occurrences, thus almost performing maximum likelihood learning of the table.
- $\mathrm{P}(\lambda_B|B, B') = 1.0\ iff\ B = B'$ is just a coherence constraint.
- $\mathrm{P}(AD, GD, ID|H)$ is a covariance table of the air, ground, invisible defense values depending on how the attack happens. As for $\mathrm{P}(E, T, TA, B|A)$, we use a Laplace's law of succession to learn it.

- $\mathrm{P}(H|P)$ is the distribution on how the attack happens depending on what is possible. Trivially $\mathrm{P}(H = ground|P = ground) = 1.0$, for more complex possibilities we have different maximum likelihood multinomial distributions on H values depending on P.
- $\mathrm{P}(P|TT)$ is the direct mapping of what the tech tree allows as possible attack types: $\mathrm{P}(P = p|TT) = 1$ is a function of TT (all $\mathrm{P}(P \neq p|TT) = 0$).
- $\mathrm{P}(TT)$: if we are sure of the tech tree (prediction without fog of war, or in decision-making mode), $\mathrm{P}(TT = k) = 1$ and $\mathrm{P}(TT \neq k) = 0$; otherwise, it allows us to take uncertainty about the opponent's tech tree and balance $\mathrm{P}(P|TT)$. We obtain a distribution on what is possible ($\mathrm{P}(P)$) for the opponent's attack types.

There are two approaches to fill up these probability tables, either by observing games (supervised learning), as we did in the evaluation section, or by acting (reinforcement learning). In match situation against a given opponent, for inputs that we can unequivocally attribute to their intention (style and general strategy), we also refine these probability tables (with Laplace's rule of succession). To keep things simple, we just refine $\sum_{E,T,TA} \mathrm{P}(E, T, TA, B|A)$ corresponding to their aggressiveness (aggro) or our successes and failures, and equivalently for $\mathrm{P}(H|P)$. Indeed, if we sum over E, T and TA, we consider the inclination of our opponent to venture into enemy territory or the interest that we have to do so by counting our successes with aggressive or defensive parameters. In $\mathrm{P}(H|P)$, we are learning the opponent's inclination for particular types of tactics according to what is available to their, or for us the effectiveness of our attack types choices.

The model is highly modular, and some parts are more important than others. We can separate three main parts: $\mathrm{P}(E, T, TA, B|A)$, $\mathrm{P}(AD, GD, ID|H)$ and $\mathrm{P}(H|P)$. In prediction, $\mathrm{P}(E, T, TA, B|A)$ uses the inferred (uncertain) economic (E), tactical (T) and belonging (B) scores of the opponent while knowing our own tactical position fully (TA). In decision-making, we know E, T, B (for us) and estimate TA. In our prediction benchmarks, $\mathrm{P}(AD, GD, ID|H)$ has the lesser impact on the results of the three main parts, either because the uncertainty from the attacker on AD, GD, ID is too high or because our heuristics are too simple, though it still contributes positively to the score. In decision-making, it allows for reinforcement learning to have pivoting tuple values for AD, GD, ID at which to switch attack types. In prediction, $\mathrm{P}(H|P)$ is used to take $\mathrm{P}(TT)$ (coming from strategy prediction Synnaeve and Bessière [2011]) into account and constraints H to what is possible. For the use of $\mathrm{P}(H|P).\mathrm{P}(P|TT).\mathrm{P}(TT)$ in decision-making, see the Results sections.

Questions

For a given region i, we can ask the probability to attack here,

$$\mathrm{P}(A_i = a_i|e_i, t_i, ta_i, \lambda_{B,i} = 1)$$

$$= \frac{\sum_{B_i,B_i'} \mathrm{P}(e_i, t_i, ta_i, B_i|a_i).\mathrm{P}(a_i).\mathrm{P}(B_i').P(\lambda_{B,i}|B_i, B_i')}{\sum_{A_i,B_i,B_i'} \mathrm{P}(e_i, t_i, ta_i, B_i|A_i).\mathrm{P}(A_i).\mathrm{P}(B_i').\mathrm{P}(\lambda_{B,i}|B_i, B_i')}$$
$$\propto \sum_{B_i,B_i'} \mathrm{P}(e_i, t_i, ta_i, B_i|a_i).\mathrm{P}(a_i).\mathrm{P}(B_i').\mathrm{P}(\lambda_{B,i}|B_i, B_i')$$

and the mean by which we should attack,

$$\mathrm{P}(H_i = h_i|ad_i, gd_i, id_i)$$
$$\propto \sum_{TT,P} [\mathrm{P}(ad_i, gd_i, id_i|h_i).\mathrm{P}(h_i|P).\mathrm{P}(P|TT).\mathrm{P}(TT)]$$

For clarity, we omitted some variables couples on which we have to sum (to take uncertainty into account) as for B (and B') above. We always sum over estimated, inferred variables, while we know

the one we observe fully. In prediction mode, we sum oover TA, B, TT, P; in decision-making, we sum over E, T, B, AD, GD, ID. The complete question that we ask our model is $\mathrm{P}(A, H|FullyObserved)$. The maximum of $\mathrm{P}(A, H)$ may not be the same as the maximum of $\mathrm{P}(A)$ or $\mathrm{P}(H)$, for instance think of a very important economic zone that is very well defended, it may be the maximum of $\mathrm{P}(A)$, but not once we take $\mathrm{P}(H)$ into account. Inversely, some regions are not defended against anything at all but present little or no interest. Our joint distribution (1) can be rewritten: $\mathrm{P}(Searched, FullyObserved, Estimated)$, so we ask:

$$\begin{aligned} & \mathrm{P}(A_{1:n}, H_{1:n}|FullyObserved) \qquad (2) \\ \propto & \sum_{Estimated} \mathrm{P}(A_{1:n}, H_{1:n}, Estimated, FullyObserved) \end{aligned}$$

6.4 Results on StarCraft

Learning

To measure fairly the prediction performance of such a model, we applied "leave-100-out" cross-validation from our dataset: as we had many games (see Table. 6.1), we set aside 100 games of each match-up for testing (with more than 1 battle per match: rather $\approxeq$ 15 battles/match) and train our model on the rest. We write match-ups XvY with X and Y the first letters of the factions involved (Protoss, Terran, Zerg). Note that mirror match-ups (PvP, TvT, ZvZ) have less games but twice as much attacks from a given faction. Learning was performed as explained in III.B.3: for each battle in r we had one observation for: $\mathrm{P}(e_r, t_r, ta_r, b_r|A = true)$, and $\#regions - 1$ observations for the i regions which were not attacked: $\mathrm{P}(e_{i\neq r}, t_{i\neq r}, ta_{i\neq r}, b_{i\neq r}|A = false)$. For each battle of type t we had one observation for $P(ad, gd, id|H = t)$ and $P(H = t|p)$. By learning with a Laplace's law of succession **?**, we allow for unseen event to have a non-null probability.

An exhaustive presentation of the learned tables is out of the scope of this paper, but we displayed interesting cases in which the learned probability tables meet concur with human expertise in Figures 6.3,6.4,6.5. In Fig. 6.3, we see that air raids/attacks are quite risk averse and it is two times more likely to attack a region with less than 1/10th of the flying force in anti-aircraft warfare than to attack a region with up to one half of our force. We can also notice than drops are to be preferred either when it is safe to land (no anti-aircraft defense) or when there is a large defense (harassment tactics). In Fig. 6.4 we can see that, in general, there are as much ground attacks at the sum of other types. The two top graphs show cases in which the tech of the attacker was very specialized, and, in such cases, the specificity seems to be used. In particular, the top right graphic may be corresponding to a "fast Dark Templars rush". Finally, Fig. 6.5 shows the transition between two types of encounters: tactics aimed at engaging the enemy army (a higher T value entails a higher $\mathrm{P}(A)$) and tactics aimed at damaging the enemy economy (at high E, we look for opportunities to attack with a small army where T is lower).

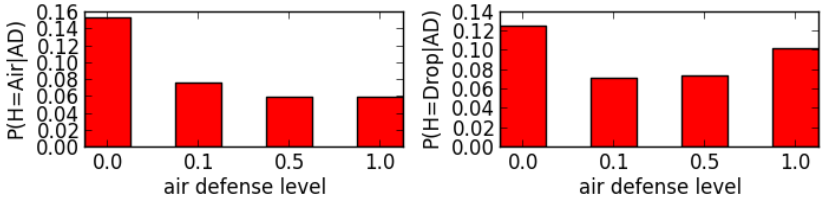


Figure 6.3: $\mathrm{P}(H = air)$ and $\mathrm{P}(H = drop)$ for varying values of AD (summed on other variables), for Terran in TvP.

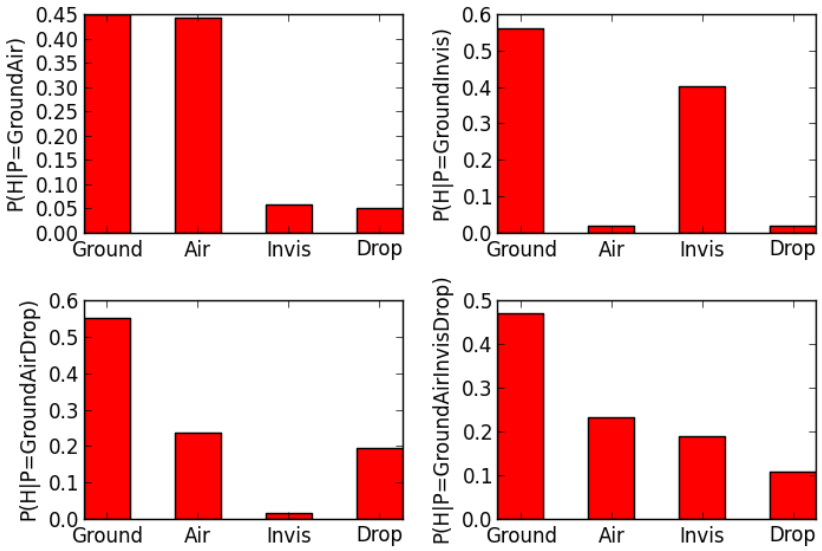


Figure 6.4: P($H|P$) for varying values H and for different values of P (derived from inferred TT), for Protoss in PvT.

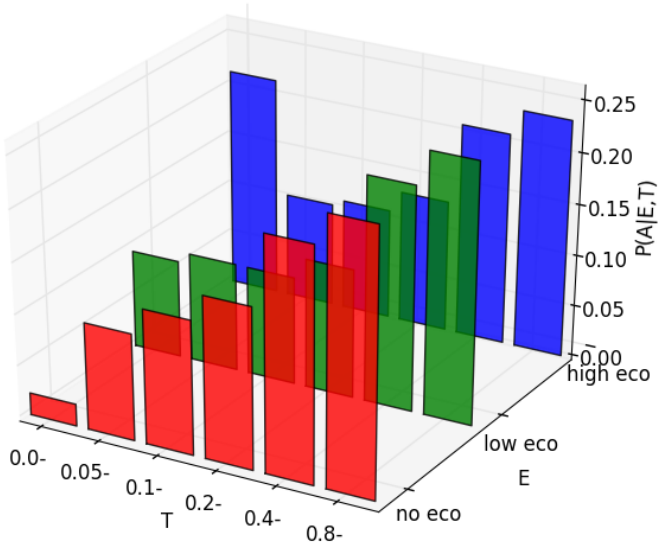


Figure 6.5: P(A) for varying values of E and T, summed on the other variables, for Terran in TvT.

Prediction Performance

We learned and tested one model for each race and each match-up. As we want to predict *where* ($\mathrm{P}(A_{1:n})$) and *how* ($\mathrm{P}(H_{battle})$) the next attack will happen to us, we used inferred enemy TT (to produce P) and TA, our scores being fully known: E, T, B, ID. We consider GD, AD to be fully known even though they depend on the attacker force, we should have some uncertainty on them, but we tested that they accounted (being known instead of fully unknown) for 1 to 2% of $\mathrm{P}(H)$ accuracy (in prediction) once P was known. We should point that pro-gamers scout very well and so it allows for a highly accurate TT estimation with Synnaeve and Bessière [2011]. Training requires to recreate battle states (all units positions) and count parameters for 5,000 to 30,000 battles. Once that is done, inference is very quick: a look-up in a probability table for known values and $\#F$ look-ups for free variables F on which we sum. We chose to try and predict the next battle 30 seconds before it happens, 30 seconds being an approximation of the time needed to go from the middle of a map (where the entropy on "next battle position" is maximum) to any region by ground, so that the prediction is useful for the defender (they can position their army).

The model code[6] (for learning and testing) as well as the datasets (see above) are freely avaible. Raw results of predictions of positions and types of attacks 30 seconds before they happen are presented in Table. 6.1: for instance the bold number (38.0) corresponds to the percentage of good positions (regions) predictions (30 sec before event) which were ranked 1st in the probabilities on $A_{1:n}$ for Protoss attacks against Terran (PvT). The measures on *where* corresponds to the percentage of good prediction and the mean probability for given ranks in $\mathrm{P}(A_{1:n})$ (to give a sense of the shape of the distribution). As the most probable The measures on *how* corresponds to the percentage of good predictions for the most probable $\mathrm{P}(H_{battle})$ and the number of such battles seen in the test set for given attack types. We particularly predict well ground attacks (trivial in the early game, less in the end game) and, interestingly, Terran and Zerg drop attacks. The *where & how* row corresponds to the percentage of good predictions for the maximal probability in the joint $\mathrm{P}(A_{1:n}, H_{1:n})$: considering only the most probable attack (more information is in the rest of the distribution, as shown for *where*!) according to our model, we can predict *where* **and** *how* an attack will occur in the next 30 seconds $\approxeq 1/4$th of the time. Finally, note that scores are not ridiculous 60 seconds before the attack neither (obviously, TT, and thus P, are not so different, nor are B and E): PvT *where* top 4 ranks are 35.6, 8.5, 7.7, 7.0% good versus 38.0, 16.3, 8.9, 6.7% 30 seconds before; *how* total precision 60 seconds before is 70.0% vs. 72.4%, *where & how* maximum probability precision is 19.9% vs. 23%.

Table 6.1: Results summary for multiple metrics at 30 seconds before attack. The number in bold (38.0) is read as "38% of the time, the region i with probability of rank 1 in $\mathrm{P}(A_i)$ is the one in which the attack happened 30 seconds later".

%: good predictions		Protoss						Terran						Zerg					
Pr: mean probability		P		T		Z		P		T		Z		P		T		Z	
total # games		1336		7225		6082		7225		1384		6322		6082		6322		598	
measure	rank	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr	%	Pr
where	1	40.9	.334	**38.0**	.329	34.5	.304	35.3	.299	34.4	.295	39.0	0.358	32.8	.31	39.8	.331	37.2	.324
	2	14.6	.157	16.3	.149	13.0	.152	14.3	.148	14.7	.0147	17.8	.174	15.4	.166	16.6	.148	16.9	.157
	3	7.8	.089	8.9	.085	6.9	.092	9.8	.09	8.4	.087	10.0	.096	11.3	.099	7.6	.084	10.7	.100
	4	7.6	.062	6.7	.059	7.9	.064	8.6	.071	6.9	.063	7.0	.062	8.9	.07	7.7	.064	8.6	.07
measure	type	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N	%	N
how	G	97.5	1016	98.1	1458	98.4	568	100	691	99.9	3218	76.7	695	86.6	612	99.8	567	67.2	607
	A	44.4	81	34.5	415	46.8	190	40	5	13.3	444	47.1	402	14.2	155	15.8	19	74.2	586
	I	22.7	225	49.6	337	12.9	132	NA	NA	NA	NA	36.8	326	32.6	227	NA	NA	NA	NA
	D	55.9	340	42.2	464	45.2	93	93.5	107	86	1183	62.8	739	67.7	535	81.4	86	63.6	588
total		76.3	1662	72.4	2674	71.9	983	98.4	806	88.5	4850	60.4	2162	64.6	1529	94.7	674	67.6	1802
where & how (%)		32.8		23		23.8		27.1		23.6		30.2		23.3		30.9		26.4	

[6] https://github.com/SnippyHolloW/AnalyzeBWData

When we are mistaken, the mean ground distance (pathfinding wise) of the most probable predicted region to the good one (where the attack happens) is 1223 pixels (38 build tiles, or 2 screens in StarCraft's resolution), while the mean max distance on the map is 5506 (172 build tiles). Also, the mean number of regions by map is 19, so a random *where* (attack destination) picking policy would have a correctness of 1/19 (5.23%). For choke-centered regions, the numbers of good *where* predictions are lower (between 24% and 32% correct for the most probable) but the mean number of regions by map is 42. For *where & how*, a random policy would have a precision of 1/(19*4), and even a random policy taking the high frequency of ground attacks into account would at most be $\approxeq$ 1/(19*2) correct. Note that our current model consider a uniform prior on regions (no bias towards past battlefields) and that we do not incorporate any derivative of the armies' movements. There is no player modeling at all: learning and fitting the mean player's tactics is not optimal, so we should specialize the probability tables for each player. Also, we use all types of battles in our training and testing. Short experiments showed that if we used only attacks on bases, the probability of good *where* predictions for the maximum of $\mathrm{P}(A_{1:n})$ goes above 50% (which is not a surprise, there are far less bases than regions in which attacks happen). To conclude on tactics positions prediction: if we sum the 2 most probable regions for the attack, we are right at least half the time; if we sum the 4 most probable (for our robotic player, it means it prepares against attacks in 4 regions as opposed to 19), we are right $\approxeq$ **70%** of the time.

Mistakes on the type of the attack are high for invisible attacks: while these tactics can definitely win a game, the counter is strategic (it is to have detectors technology deployed) more than positional. Also, if the maximum of $\mathrm{P}(H_{battle})$ is wrong, it doesn't mean than $\mathrm{P}(H_{battle} = good) = 0.0$ at all! The result needing improvements the most is for air tactics, because countering them really is positional, see our discussion in the conclusion.

In Game Decision-Making

In a StarCraft game, our bot has to make decisions about where and how to attack or defend, it does so by reasoning about opponent's tactics, bases, its priors, and under strategic constraints (Fig. **??**). Once a decision is taken, the output of the tactical model is an offensive or defensive goal. There are different military goal types (base defense, ground attacks, air attacks, drops...), and each type of goal has pre-requisites (for instance: a drop goal needs to have the control of a dropship and military units to become active). The spawned goal then autonomously sets objectives for Bayesian units Synnaeve and Bessière [2011a], sometimes procedurally creating intermediate objectives or canceling itself in the worst cases.

The destinations of goals are from $\mathrm{P}(A)$, while the type of the goal comes from $\mathrm{P}(H)$. In input, we fully know tactical scores of the regions according to our military units placement TA (we are the attacker), what is possible for us to do P (according to units available) and we estimate E, T, B, ID, GD, AD from past (partial) observations. Estimating T is the most tricky of all because it may be changing fast, for that we use a units filter which just decays probability mass of seen units. An improvement would be to use a particle filter **?**, with a learned motion model. From the joint (2) $\mathrm{P}(A_{1:n}, H_{1:n}|ta, p, tt)$ may arise a couple i, H_i more probable than the most probables $\mathrm{P}(A_i)$ and $\mathrm{P}(H_j)$ taken separately (the case of an heavily defended main base and a small unprotected expand for instance). Fig. 6.6 displays the mean $\mathrm{P}(A, H)$ for Terran (in TvZ) attacks decision-making for the most 32 probable type/region tactical couples. It is in this kind of landscape (though more steep because Fig. 6.6 is a mean) that we sample (or pick the most probable couple) to take a decision. Also, we may spawn defensive goals countering the attacks that we predict from the opponent.

Finally, we can steer our technological growth towards the opponent's weaknesses. A question that we can ask our model (at time t) is $\mathrm{P}(TT)$, or, in two parts: we first find i, h_i which maximize $\mathrm{P}(A, H)$ at time $t + 1$, and then ask a more directive:

$$\mathrm{P}(TT|h_i) \propto \sum_P \mathrm{P}(h_i|P).\mathrm{P}(P|TT).\mathrm{P}(TT)$$

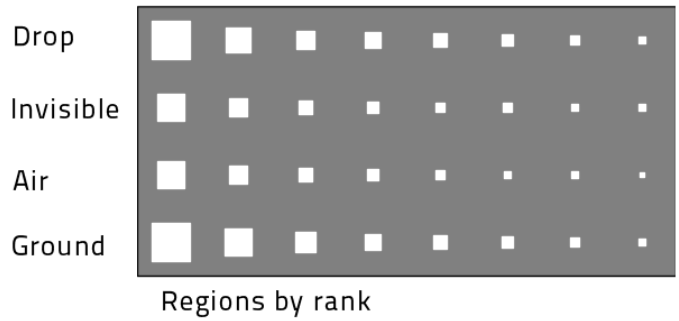


Figure 6.6: Mean $\mathrm{P}(A, H)$ for all H values and the top 8 $\mathrm{P}(A_i, H_i)$ values, for Terran in TvZ. The larger the white square area, the higher $\mathrm{P}(A_i, H_i)$.

so that it gives us a distribution on the tech trees (TT) needed to be able to perform the wanted attack type. To take a decision on our technology direction, we can consider the distances between our current tt^t and all the probable values of TT^{t+1}.

6.5 Extensions

There are three main research directions for possible improvements: improving the underlying heuristics, improving the dynamic of the model and improving the model itself. The heuristics presented here are quite simple but they may be changed, and even removed or added, for another RTS or FPS, or for more performance. In particular, our "defense against invisible" heuristic could take detector positioning/coverage into account. Our heuristic on tactical values can also be reworked to take terrain tactical values into account (chokes and elevation in StarCraft). For the estimated position of enemy units, we could use a particle filter **?** with a motion model (at least one for ground units and one for flying units). There is room to improve the dynamics of the model: considering the prior probabilities to attack in regions given past attacks and/or considering evolutions of the T,TA,B,E values (derivatives) in time. The discretizations that we used may show their limits, though if we want to use continuous values, we need to setup a more complicated learning and inference process (MCMC sampling). Finally, one of the strongest assumptions (which is a drawback particularly for prediction) of our model is that the attacking player is always considered to attack in this most probable regions. While this would be true if the model was complete (with finer army positions inputs and a model of what the player thinks), we believe such an assumption of completeness is far fetched. Instead we should express that incompleteness in the model itself and have a "player decision" variable $D \sim Multinomial(\mathrm{P}(A_{1:n}, H_{1:n}), player)$.

Chapter 7

Strategy

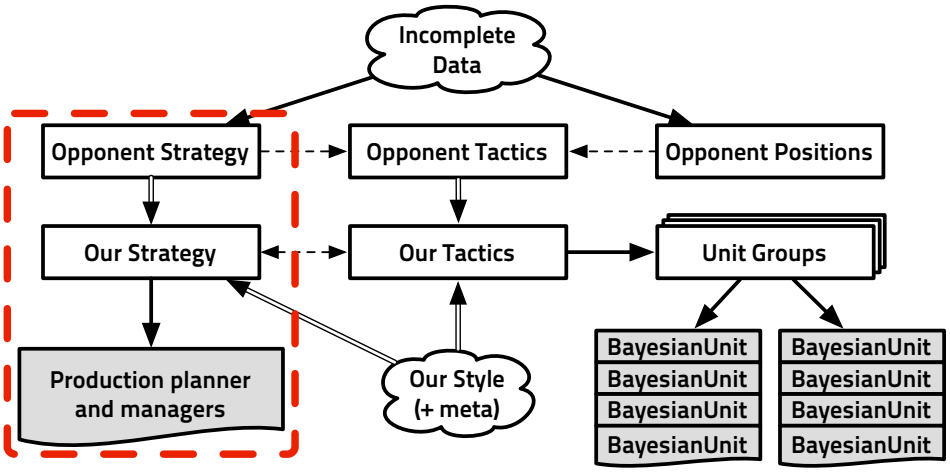


Figure 7.1: Information-centric view of the architecture of the bot, the part concerning this chapter is in the dotted rectangle

- Problem: take the winning strategy in the absolute
- Complexity: without going meta, prediction is an "inference under uncertainty" problem, adaptation w.r.t. what we know is a planning under constraints problem. Going meta ⇒ `(O.O)`
- State of the art: Data mining, plan recognition, CBR... [Weber and Mateas, 2009], [Weber and Ontañón, 2010] and even meta [Manish Meta, 2010].
- Our take: [Synnaeve and Bessière, 2011b], [Synnaeve and Bessière, 2011]
- Results: better resistance to noise, which is fundamental in real setup RTS gameplay (fog of war). Full Bayesian model down to adaptation actions some day?
- Conclusion and perspectives: a way to encode and use gameplay/structural knowledge

7.1 What is a Strategy?

In a RTS, players need to gather resources to build military units and defeat their opponents. To that end, they often have *worker units* (or extraction structures) than can gather resources needed to build

workers, *buildings*, *military units* and *research upgrades*. Workers are often also builders (as in StarCraft) and are weak in fights compared to military units. Resources may have different uses, for instance in StarCraft: minerals are used for everything, whereas gas is only required for advanced buildings or military units, and technology upgrades. Buildings and research upgrades define technology trees (directed acyclic graphs) and each state of a "tech tree" allow for different unit type production abilities and unit spells/abilities. The military can be of different types, any combinations of ranged, casters, contact attack, zone attacks, big, small, slow, fast, invisible, flying... Units can have attacks and defenses that counter each others as in rock-paper-scissors.

Each unit and building has a *sight range* that provides the player with a view of the map. Parts of the map not in the sight range of the player's units are under *fog of war* and the player ignores what is and happens there. In RTS games jargon, an *opening** denotes the same thing than in Chess: an early game plan for which the player has to make choices. In Chess because one can not move many pieces at once (each turn), in RTS games because during the development phase, one is economically limited and has to choose between economic and military priorities and can only open so many tech paths at once. The *opening** corresponds to the first military (tactical) moves that will be performed and, in StarCraft, it corresponds to the 5 (early rushes) to 15 minutes (advanced technology / late push) timespan. Players have to find out what opening their opponents are doing to be able to effectively deal with the strategy (army composition) and tactics (military moves: where and when) thrown at them. For that, players scout each other and reason about the incomplete information they can bring together about army and buildings composition. This paper presents a probabilistic model able to predict the *opening** of the enemy that is used in a StarCraft AI competition entry bot (see Figure **??**).

7.2 Strategy prediction

Related Works

This work was encouraged by the reading of Weber and Mateas' Data Mining Approach to Strategy Prediction [Weber and Mateas, 2009] and the fact that they provided their dataset, that we used. They tried and evaluated several machine learning algorithms on replays that were labeled with strategies (openings) with rules.

There are related works in the domains of opponent modeling [Hsieh and Sun, 2008, Schadd et al., 2007, **?**]. The main methods used to these ends are case-based reasoning (CBR) and planning or plan recognition [Aha et al., 2005, **?**, **?**, **?**, Ramírez and Geffner, 2009]. There are precedent works of Bayesian plan recognition [**?**], even in games with Albrecht *et al.* [**?**] using dynamic Bayesian networks to recognize a user's plan in a multi-player dungeon adventure.

Aha *et al.* [Aha et al., 2005] used CBR to perform dynamic plan retrieval extracted from domain knowledge in Wargus (Warcraft II clone). Ontañón *et al.* [**?**] base their real-time case-based planning (CBP) system on a plan dependency graph which is learned from human demonstration. In [**??**], they use CBR and expert demonstrations on Wargus. They improve the speed of CPB by using a decision tree to select relevant features. Hsieh and Sun [Hsieh and Sun, 2008] based their work on Aha *et al.*'s CBR [Aha et al., 2005] and used StarCraft replays to construct states and building sequences. Strategies are choices of building construction order in their model.

Schadd *et al.* [Schadd et al., 2007] describe opponent modeling through hierarchically structured models of the opponent behaviour and they applied their work to the Spring RTS (Total Annihilation clone). Hoang *et al.* [**?**] use hierarchical task networks (HTN) to model strategies in a first person shooter with the goal to use HTN planners. Kabanza *et al.* [**?**] improve the probabilistic hostile agent task tracker (PHATT [**?**], a simulated HMM for plan recognition) by encoding strategies as HTN.

The work described in this section can be classified as probabilistic plan recognition. Strictly speaking, we present model-based machine learning used for prediction of plans, while our model is not limited to prediction. It performs two levels of plan recognition, both are learned from the replays: tech

tree prediction (unsupervised) and opening prediction (semi-supervised or supervised depending on the labeling method).

Replays Labeling

We used Weber and Mateas [Weber and Mateas, 2009] dataset of labeled replays. It is composed of 9316 StarCraft: Broodwar game logs, between ≈ 500 and 1300 per *match-up*. A match-up is a set of the two opponents races, Protoss versus Terran (PvT) is a match-up, PvZ is another one. They are distinguished because strategies distribution are very different across match-ups (see Table 7.2). Weber and Mateas used logic rules on building sequences to put their labels, concerning only tier 2 strategies (no tier 1 rushes).

Openings are closely related to *build orders* (BO) but different BO can lead to the same opening and some BO are shared by different openings. Particularly, if we do not take the time at which the buildings are constructed, we may be wrong too often. For that reason, we tried to label replays with the statistical appearance of key features with a semi-supervised method (see Figure 7.2). Indeed, the purpose of our opening prediction model is to help our StarCraft playing bot to deal with rushes and special tactics. This was not the main focus of Weber and Mateas' labels, which follow more the development of the tech tree. We used the key components of openings that we want to be aware of as features for our labeling algorithm as show in Table 7.1.

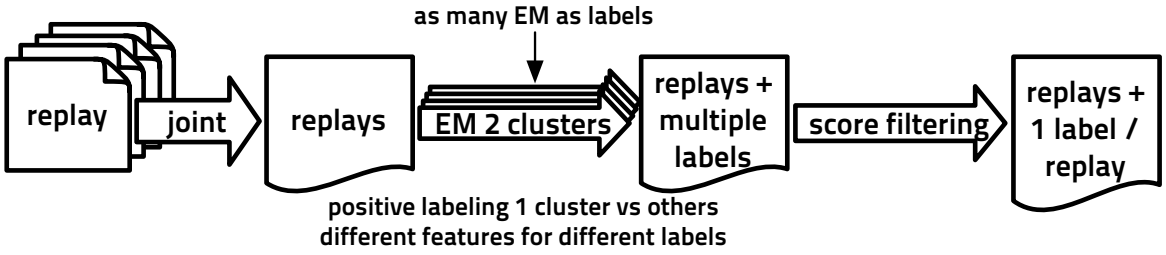


Figure 7.2: Data centric view of our semi-supervised labeling of replays

The selection of the features along with the opening labels is the supervised part of our labeling method. The knowledge of the features and openings comes from expert play and the StarCraft liquipedia[1]. They are all presented in Table 7.1. For instance, if we want to find out which replays correspond to the "fast Dark Templar" (DT, Protoss invisible unit) opening, we put the time at which the first Dark Templar is constructed as a feature and perform clustering on replays with it. This is what is needed for our playing bot: to be able to know when he has to fear "fast DT" opening and build a detector unit quickly to be able to deal with invisibility.

For the clustering part, we tried k-means, expectation-maximization (EM) with equal shape (bivariate normal distribution with proportional covariances matrices) and EM with the normal distribution shapes and volumes chosen with a Bayesian information criterion (BIC). Best BIC models were almost always the most agreeing with expert knowledge (15/17 labels). We used the R package Mclust [**??**] to perform full EM clustering. We produce "2 bins clustering" for each set of features (corresponding to each opening), and label the replays belonging to the cluster with the lower norm of features' appearances (that is exactly the purpose of our features). Figures 7.5 and 7.6 show the clusters out of EM with the features of the corresponding openings. We thought of clustering because there are two cases in which you build a specific military unit of research a specific upgrade: either it is part of your opening, or it is part of your longer term game plan or even in reaction to the opponent. So the distribution over

[1] `http://wiki.teamliquid.net/starcraft/`

the time at which a feature appears is bimodal, with one (sharp) mode corresponding to "opening with it" and the other for the rest of the games, as can be seen in Figure 7.3.

Table 7.1: Opening/Strategies labels of the replays (Weber's and ours are not always corresponding)

Race	Weber's labels	Our labels	Features	Note (what we fear)
Protoss	FastLegs	speedzeal	Legs, GroundWeapons+1	quick speed+upgrade attack
	FastDT	fast_dt	DarkTemplar	invisible units
	FastObs	nony	Goon, Range	quick long ranged attack
	ReaverDrop	reaver_drop	Reaver, Shuttle	tactical attack zone damages
	Carrier	corsair	Corsair	flying units
	FastExpand	templar	Storm, Templar	powerful zone attack
		two_gates	2ndGateway, Gateway, Zealot	aggressive rush
	Unknown	unknown	(no clear label)	
Terran	Bio	bio	3rdBarracks, 2ndBarracks, Barracks	aggressive rush
	TwoFactory	two_facto	2ndFactory	strong push (long range)
	VultureHarass	vultures	Mines, Vulture	aggressive harass, invisible
	SiegeExpand	fast_exp	Expansion, Barracks	economical advantage
	Standard			
	FastDropship	drop	DropShip	tactical attack
	Unknown	unknown	(no clear label)	
Zerg	TwoHatchMuta	fast_mutas	Mutalisk, Gas	early air raid
	ThreeHatchMuta	mutas	3rdHatch, Mutalisk	massive air raid
	HydraRush	hydras	Hydra, HydraSpeed, HydraRange	quick ranged attack
	Standard	(speedlings)	(ZerglingSpeed, Zergling)	(removed, quick attacks/mobility)
	HydraMass			
	Lurker	lurkers	Lurker	invisible and zone damages
	Unknown	unknown	(no clear label)	

Table 7.2: Openings distributions for Terran in all the match-ups

	vs Protoss		vs Terran		vs Zerg	
Opening	Nb	Percentage	Nb	Percentage	Nb	Percentage
bio	62	6.2	25	4.4	197	22.6
fast_exp	438	43.5	377	65.4	392	44.9
two_facto	240	23.8	127	22.0	116	13.3
vultures	122	12.1	3	0.6	3	0.3
drop	52	5.2	10	1.7	121	13.9
unknown	93	9.3	34	5.9	43	5.0

Finally, some replays are labeled two or three times with different labels (due to the different time of effect of different openings), so we apply a filtering to transform multiple label replays into unique label ones (see Figure 7.2). For that we choose the openings labels that were happening the earliest (as they are a closer threat to the bot in a game setup) if and only if they were also the most probable or at 10% of probability of the most probable label (to exclude transition boundaries of clusters) for this replay. We find the earliest by comparing the norms of the clusters means in competition. All replays without a label or with multiple labels (*i.e.* which did not had a unique solution in filtering) after the filtering were labeled as *unknown*. We then used this labeled dataset as well as Weber and Mateas' labels in the testing of our Bayesian model for opening prediction.

Opening Prediction Model

Our predictive model is a Bayesian program, it can be seen as the "Bayesian network" represented in Figure 7.7. It is a generative model and this is of great help to deal with the parts of the observations' space where we do not have too much data (RTS games tend to diverge from one another as the number of possible actions grow exponentially). Indeed, we can model our uncertainty by putting a large standard deviation on too rare observations and generative models tend to converge with fewer observations than discriminative ones [**?**]. Here is the description of our Bayesian program:

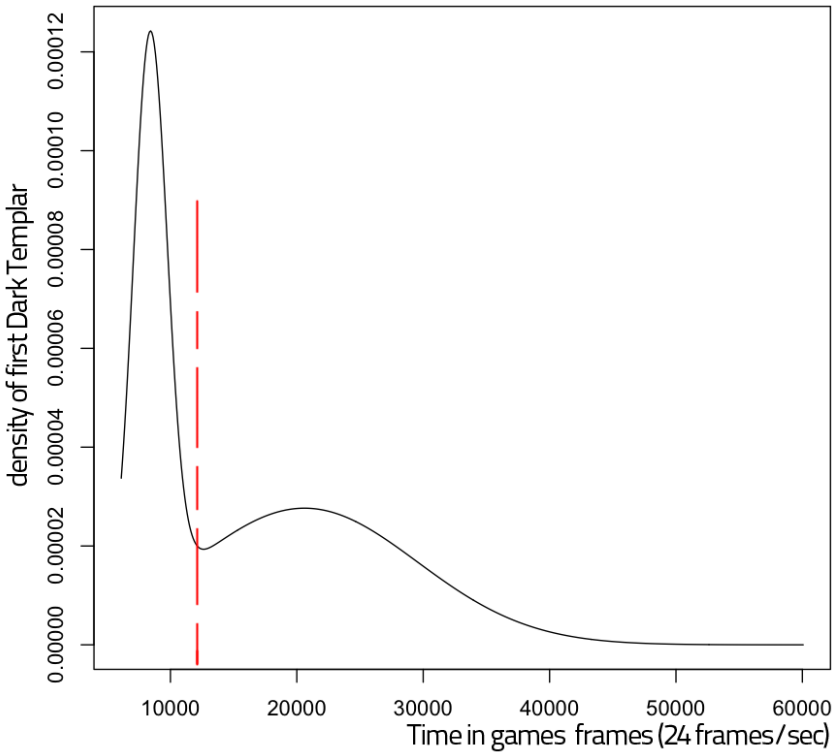


Figure 7.3: Protoss vs Terran distribution of first appearance of Dark Templars (Protoss invisible unit).

Variables

- $BuildTree \in [\emptyset, building_1, building_2, building_1 \wedge building_2, techtrees, \dots]$: all the possible building trees for the given race. For instance $\{pylon, gate\}$ and $\{pylon, gate, core\}$ are two different $BuildTrees$.

- $N\ Observations$: $O_{i \in [\![1 \dots N]\!]} \in \{0, 1\}$, O_k is 1 $(true)$ if we have seen (observed) the kth building (it can have been destroyed, it will stay "seen").

- $Opening$: $Op^t \in [opening_1 \dots opening_M]$ take the various opening values (depending on the race).

- $LastOpening$: $Op^{t-1} \in [opening_1 \dots opening_M]$, Opening value of the previous time step (allows filtering, taking previous inference into account).

- $\lambda \in \{0, 1\}$: coherence variable (restraining $BuildTree$ to possible values with regard to $O_{[\![}1 \dots N]\!]$)

- $Time$: $T \in [\![1 \dots P]\!]$, time in the game (1 second resolution).

At first, we generated all the possible (according to the game rules) $BuildTree$ values (between $\approx$ 500 and 1600 depending on the race). We observed that a lot of possible $BuildTree$ values are too absurd to be performed in a competitive match and were never seen during the learning. So, we restricted $BuildTree$ to have its value in all the build trees encountered in our replays dataset. There are 810 build trees for Terran, 346 for Protoss and 261 for Zerg ($\approx$ 3000 replays/race), all learned from the (unlabeled) replays.

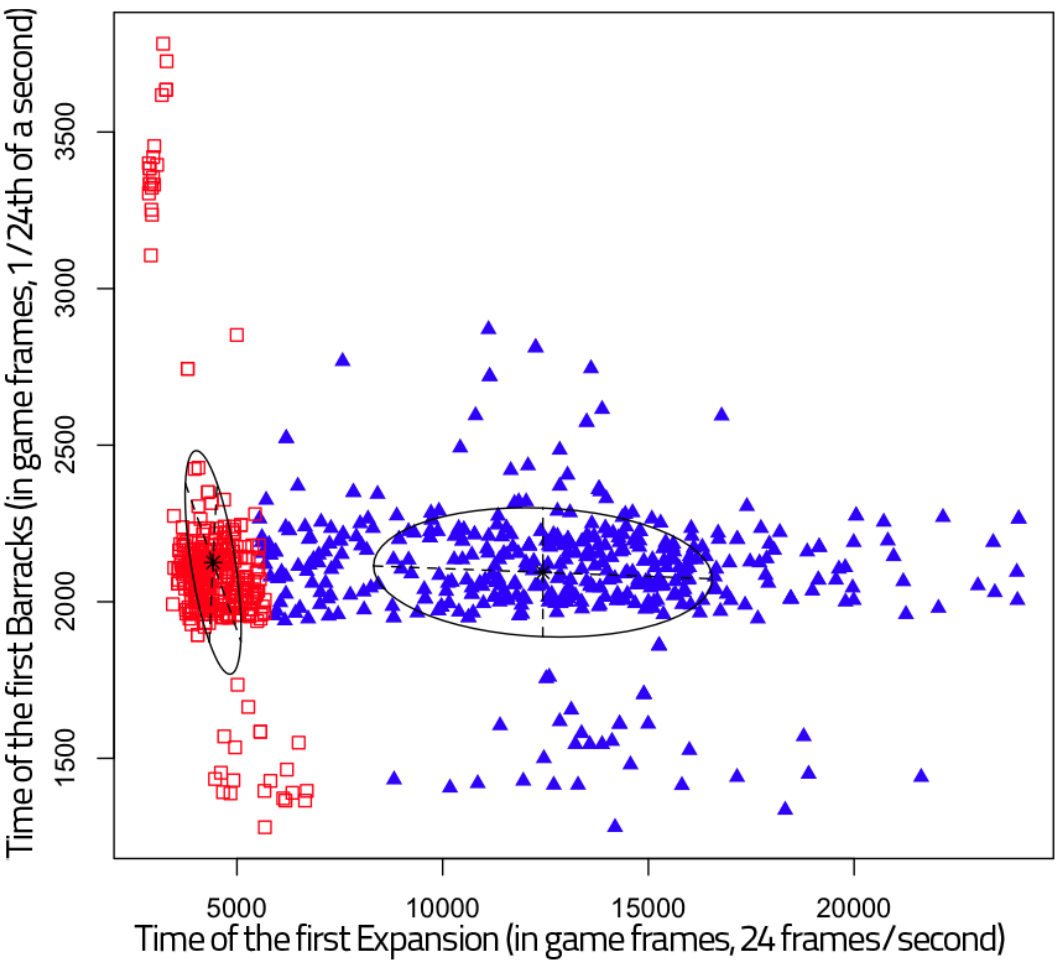


Figure 7.4: Terran vs Zerg Barracks and first Expansion timings (Terran). The bottom left cluster (squares) is the one labeled as *fast_exp*.

Decomposition

The joint distribution of our model is the following:

$$
\begin{aligned}
& P(T, BuildTree, O_1 \dots O_N, Op^t, Op^{t-1}, \lambda) \\
= \; & P(Op^t|Op^{t-1}) \\
& P(Op^{t-1}) \\
& P(BuildTree|Op^t) \\
& P(O_{[\![1\dots N]\!]}) \\
& P(\lambda|BuildTree, O_{[\![1\dots N]\!]}) \\
& P(T|BuildTree, Op^t)
\end{aligned}
$$

This can also be see as Figure 7.7.

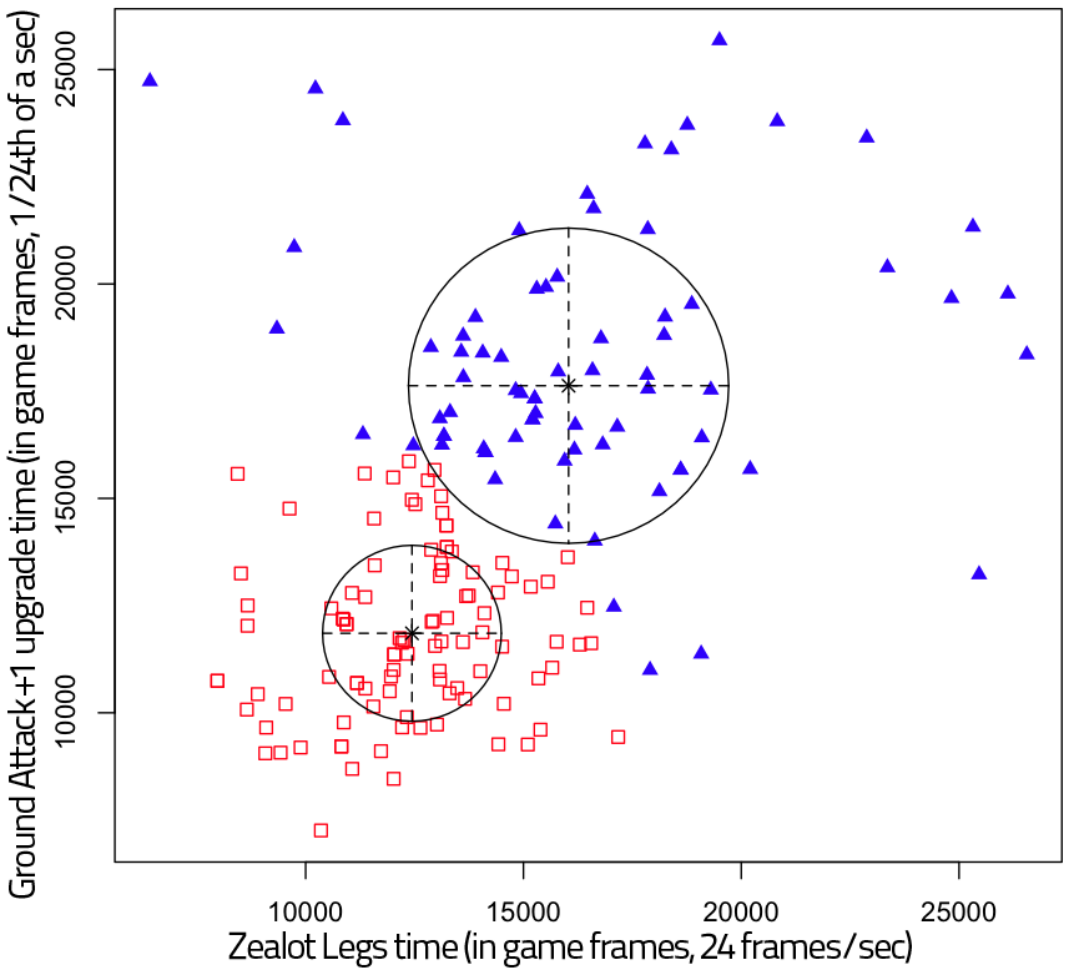


Figure 7.5: Protoss vs Protoss Ground Attack +1 and Zealot Legs upgrades timings. The bottom left cluster (squares) is the one labeled as *speedzeal*.

Forms

- $P(Op^t|Op^{t-1})$ is optional, we use it as a filter so that the previous inference impacts the current one. We use a functional Dirac:

$$\begin{aligned} & P(Op^t|Op^{t-1}) \ (Dirac) \\ = \ & 1 \text{ if } Op^t = Op^{t-1} \\ = \ & 0 \text{ else} \end{aligned}$$

  This does not prevent our model to switch predictions, it just uses previous inference posterior $P(Op^{t-1}$ to average $P(Op^t)$.

- $P(Op^{t-1})$ copied from one inference to another (mutated from $P(Op^t)$). The first $P(Op^{t-1})$ is bootstrapped with the uniform distribution, we could also use a prior on openings in the given match-up.

- $P(BuildTree|Op^t)$ is learned from the labeled replays. $P(BuildTree|Op^t)$ are $\text{card}(\{openings\})$ different histogram over the values of $BuildTree$.

- $P(O_{[\![1\ldots N]\!]})$ is unspecified, we put the uniform distribution (we could use a prior over the most frequent observations).

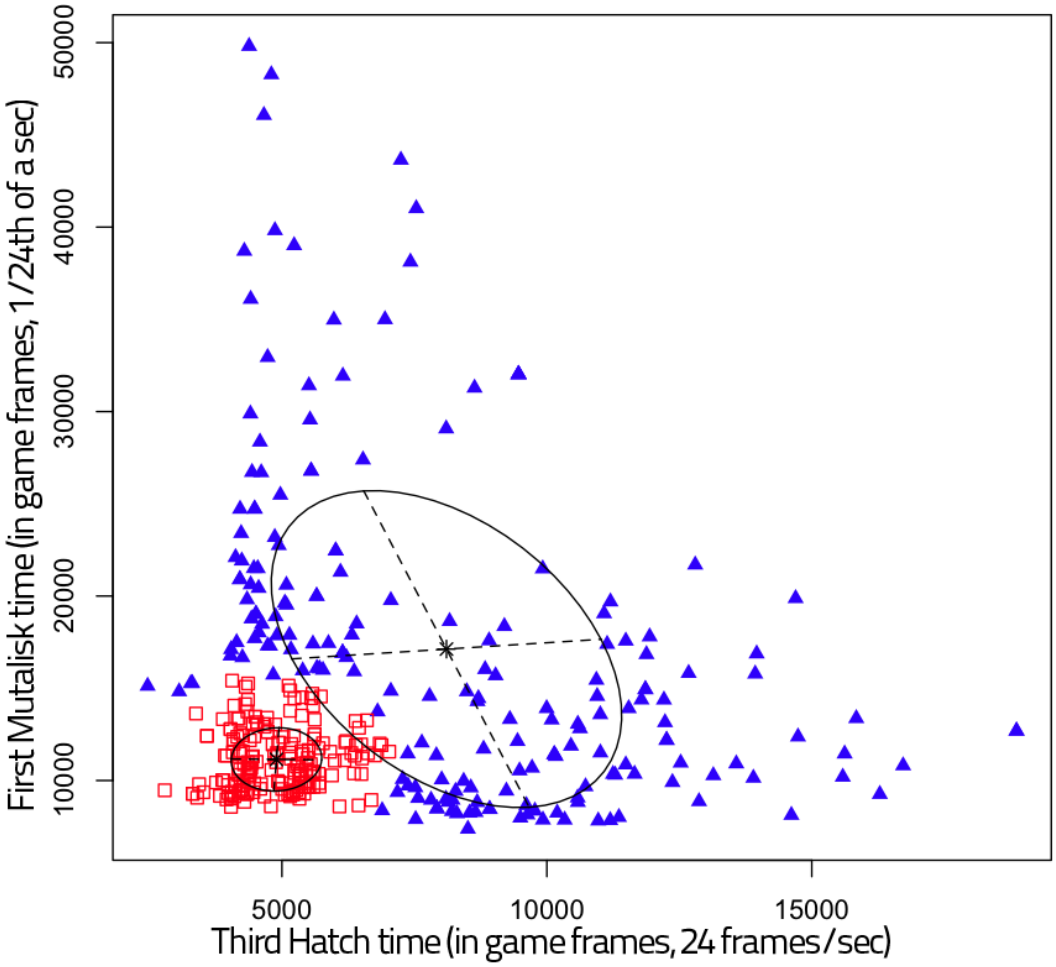


Figure 7.6: Zerg vs Protoss time of the third Hatch and first appearance of Mutalisks. The bottom left cluster (squares) is the one labeled as *mutas*.

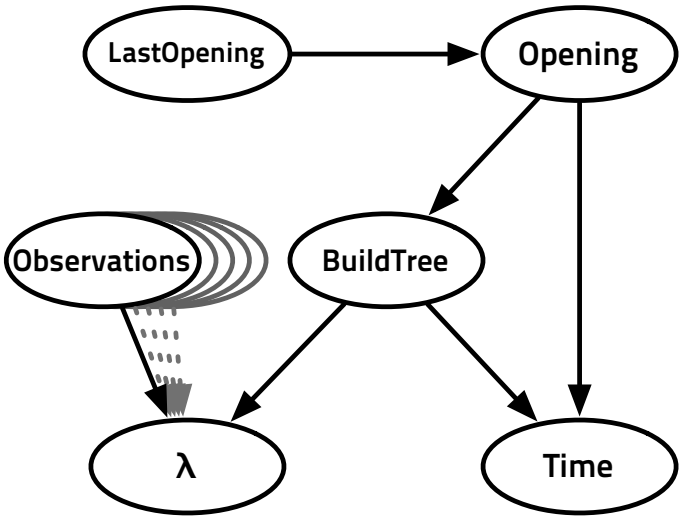


Figure 7.7: Graph representation of the opening (and tech tree) prediction model

- $P(\lambda|BuildTree, O_{[\![1...N]\!]})$ is a functional Dirac that restricts $BuildTree$ values to the ones than can co-exist with the observations.

$$\begin{aligned} & P(\lambda = 1|buildTree, o_{[\![1...N]\!]}) \\ = \; & 1 \text{ if } buildTree \text{ can exist with } o_{[\![1...N]\!]} \\ = \; & 0 \text{ else} \end{aligned}$$

  A $BuildTree$ value ($buildTree$) is compatible with the observations if it covers them fully. For instance, $BuildTree = \{pylon, gate, core\}$ is compatible with $o_{\#core} = 1$ but it is not compatible with $o_{\#forge} = 1$. In other words, $buildTree$ is incompatible with $o_{[\![1...N]\!]}$ *iff* $\{o_{[\![1...N]\!]} \backslash \{o_{[\![1...N]\!]} \wedge buildTree\}\} \neq \emptyset$.

- $P(T|BuildTree, Op^t)$ are "bell shape" distributions (discretized normal distributions). There is one bell shape per couple $(opening, buildTree)$. The parameters of these discrete Gaussian distributions are learned from the labeled replays.

Identification (learning)

The learning of the $P(BuildTree|Op^t)$ histogram is straight forward counting of occurrences from the labeled replays. The learning of the $P(T|BuildTree, Op^t)$ bell shapes parameters takes into account the uncertainty of the couples $(buildTree, opening)$ for which we have few observations. Indeed, the normal distribution $P(T|buildTree, opening)$ begins with a high σ^2, and **not** a Dirac with μ on the seen T value and $sigma = 0$. This accounts for the fact that the first observation may have been an outlier. This learning process is independent on the order of the stream of examples, seeing point A and then B or B and then A in the learning phase produces the same result.

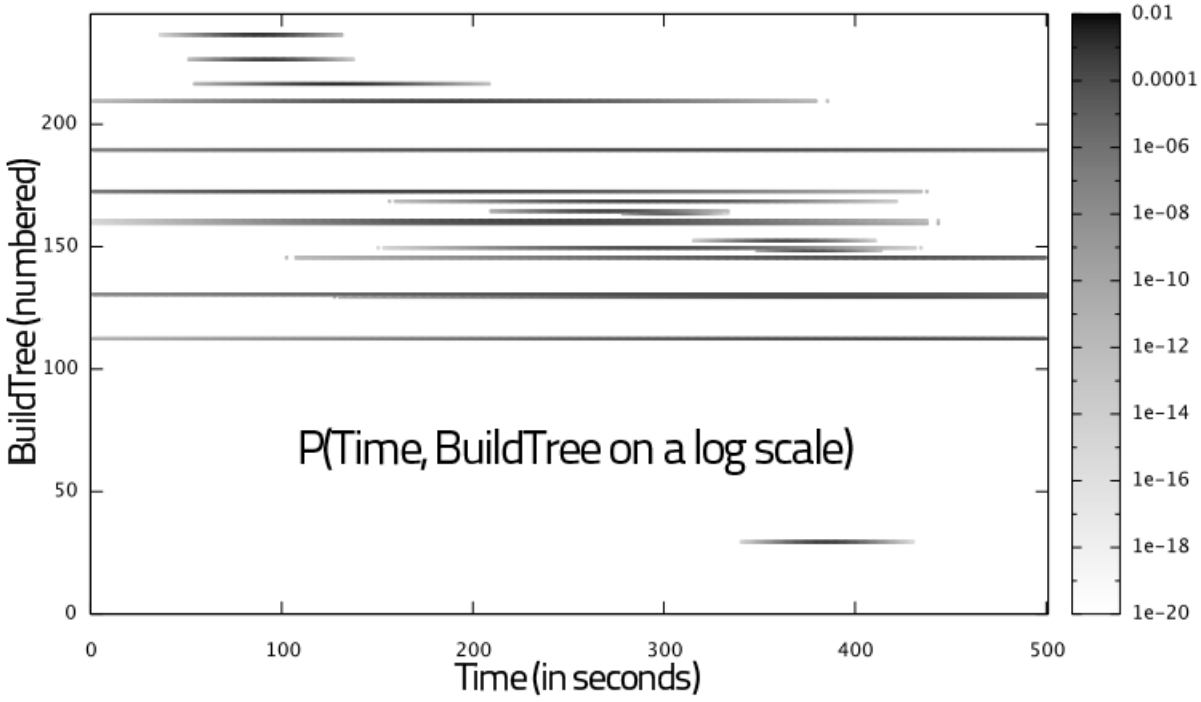


Figure 7.8: $P(Time, BuildTree|Opening^t = ReaverDrop)$

Questions

The question that we will ask in all the benchmarks is:

$$\begin{aligned} &P(Op|T=t, O_{[\![1...N]\!]}=o_{[\![1...N]\!]}, \lambda=1) \\ &\propto P(Op).P(o_{[\![1...N]\!]}) \\ &\times \textstyle\sum_{BuildTree} P(\lambda|BuildTree, o_{[\![1...N]\!]}) \\ &.P(BuildTree|Op).P(t|BuildTree, Op) \end{aligned}$$

Note that if we see $P(BuildTree, Time)$ as a plan, asking $P(BuildTree|Opening, Time)$ boils down to use our "plan recognition" mode as a planning algorithm, which could provide good approximations of the optimal goal set [Ramírez and Geffner, 2009]. This gives us a distribution on the build trees to follow (build orders) to achieve a given opening.

Results on StarCraft

Prediction

For each match-up, we ran cross-validation testing with 9/10th of the dataset used for learning and the remaining 1/10th of the dataset used for testing. We ran tests finishing at 5, 10 and 15 minutes to capture all kinds of openings (early to late ones). To measure the predictive capability of our model, we used 3 metrics:

- the *final* prediction, which is the opening that is predicted at the end of the test,
- the *online twice* (OT), which counts the openings that have emerged as most probable twice a test (so that their predominance is not due to noise),
- the *online once* > 3 (OO3), which counts the openings that have emerged as most probable openings after 3 minutes (so that these predictions are based on really meaningful information).

After 3 minutes, a Terran player will have or be building his first supply depot, barracks, refinery (gas), and at least factory or expansion. A Zerg player would have his first overlord, zergling pool, extractor (gas) and most of the time his expansion and lair tech. A Protoss player would have his first pylon, gateway, assimilator (gas), cybernectics core, and sometimes his robotics center, or forge and expansion.

Table 7.3 sums up all the prediction probabilities (scores) of our model in all the match-ups with both labeling of the game logs. Please note that when an opening is mispredicted, the distribution on openings is often not $P(badopening) = 1, P(others) = 0$ and that we can extract some value out of these distributions. Also, we observed that $P(Opening = unknown) > P(others)$ is often a case of misprediction: our bot would use the next prediction in this case. Figure 7.9 shows the evolution of the distribution $P(Opening)$ during a replay for Weber's and our labelings. Figure 7.10 shows the resistance of our model to noise. We randomly removed some observations (buildings, attributes), from 1 to 15, knowing that for Protoss and Terran we use 16 buildings observations and 17 for Zerg. We think that our model copes well with noise because it backtracks unseen observations: for instance if we have only the $core$ observation, it will work with build trees containing $core$ that will passively infer unseen $pylon$ and $gate$. Also, uncertainty is handled natively.

Performances

The first iteration of this model was not making use of the structure imposed by the game in the form of "possible build trees" and was at best very slow, at worst intractable without sampling. With the model presented here, the performances are ready for production as shown in Table 7.4. The memory footprint is around 3.5Mb on a 64bits machine. Learning computation time is linear in the number of games logs events ($O(N)$ with N observations), which are bounded, so it is linear in the number of game logs. It can

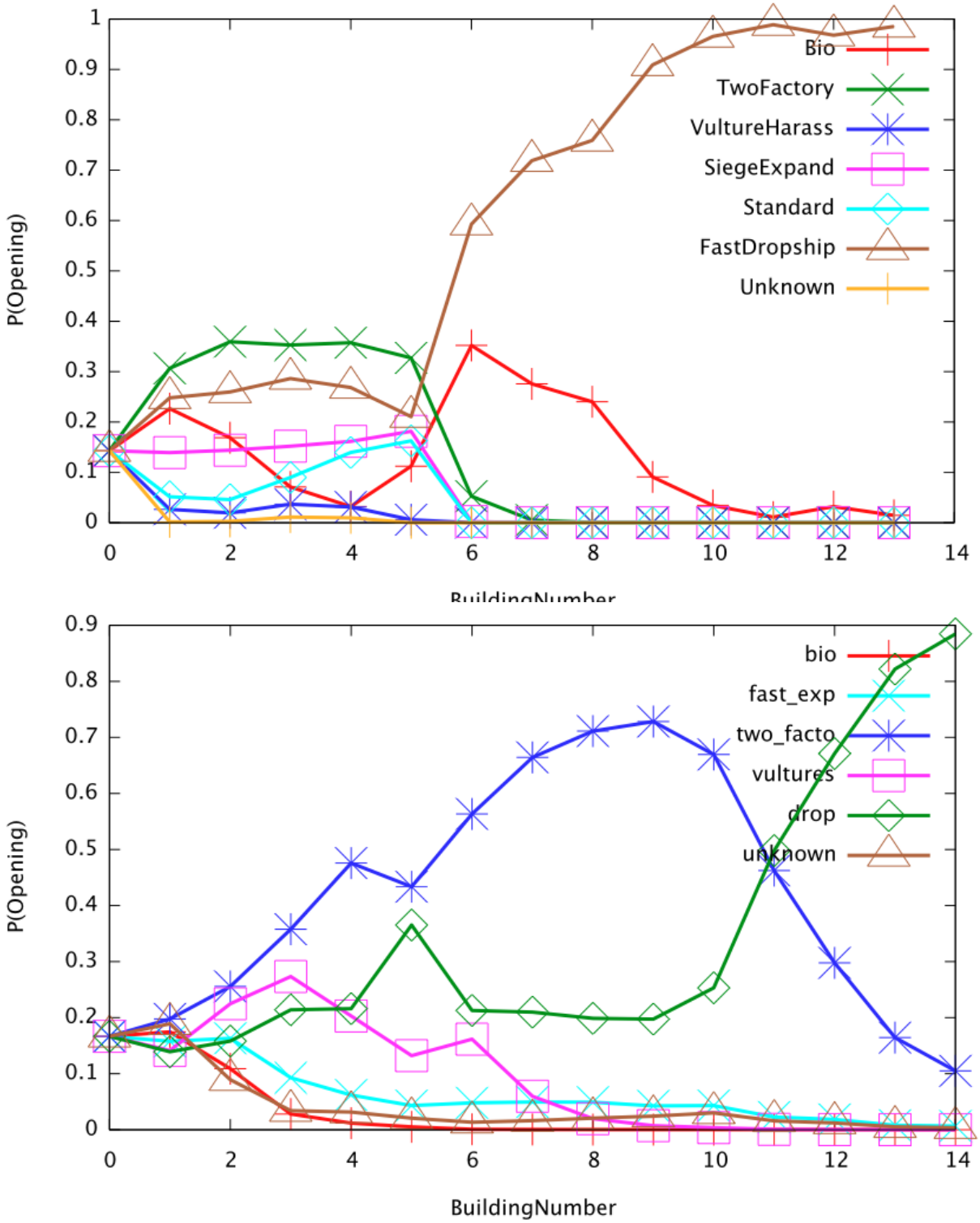
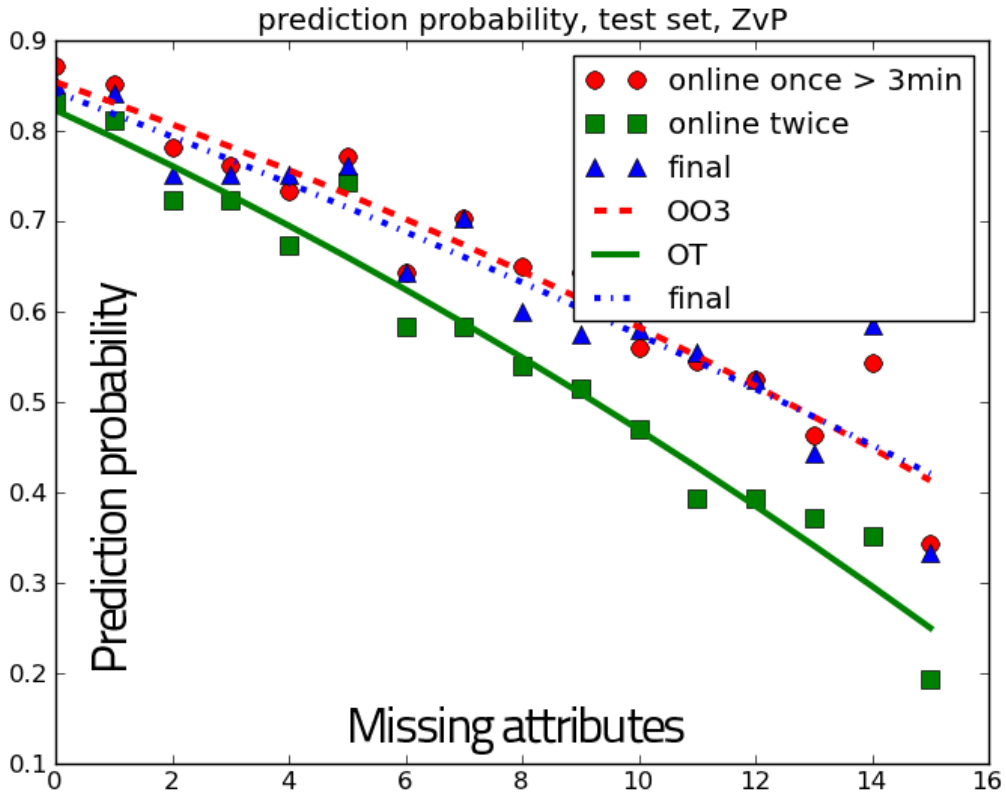


Figure 7.9: Evolution of $P(Opening)$ with increasing observations in a TvP match-up, with Weber's labeling on top, our labeling on the bottom. The x-axis corresponds to the construction of buildings.



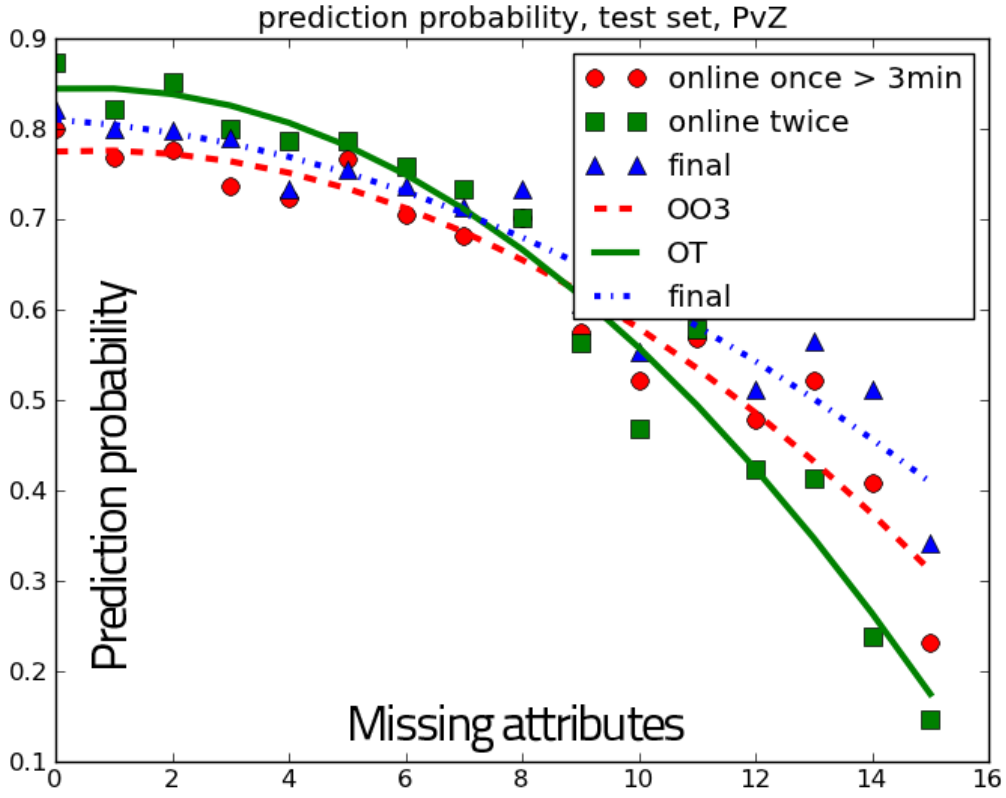


Figure 7.10: Two extreme evolutions of the 3 probabilities of opening recognition with increasing noise (15 missing attributes/observations/buildings correspond to 93.75% missing information for Protoss and Terran openings prediction and 88.23% of missing attributes for Zerg openings prediction). Zerg opening prediction probabilitly on top, Protoss bottom.

Table 7.3: Prediction probabilities for all the match-ups

	Weber and Mateas' labels									Our labels							
	5 minutes			10 minutes			15 minutes			5 minutes			10 minutes			15 minu	
match-up	final	OT	OO3	final	OT	OO3	final	OT	OO3	final	OT	OO3	final	OT	OO3	final	OT
PvP	0.65	0.53	0.59	0.69	0.69	0.71	0.65	0.67	0.73	0.78	0.74	0.68	0.83	0.83	0.83	0.85	0.83
PvT	0.75	0.64	0.71	0.78	0.86	0.83	0.81	0.88	0.84	0.62	0.69	0.69	0.62	0.73	0.72	0.6	0.79
PvZ	0.73	0.71	0.66	0.8	0.86	0.8	0.82	0.87	0.8	0.61	0.6	0.62	0.66	0.66	0.69	0.61	0.62
TvP	0.69	0.63	0.76	0.6	0.75	0.77	0.55	0.73	0.75	0.50	0.47	0.54	0.5	0.6	0.69	0.42	0.62
TvT	0.57	0.55	0.65	0.5	0.55	0.62	0.4	0.52	0.58	0.72	0.75	0.77	0.68	0.89	0.84	0.7	0.88
TvZ	0.84	0.82	0.81	0.88	0.91	0.93	0.89	0.91	0.93	0.71	0.78	0.77	0.72	0.88	0.86	0.68	0.82
ZvP	0.63	0.59	0.64	0.87	0.82	0.89	0.85	0.83	0.87	0.39	0.56	0.52	0.35	0.6	0.57	0.41	0.61
ZvT	0.59	0.51	0.59	0.68	0.69	0.72	0.57	0.68	0.7	0.54	0.63	0.61	0.52	0.67	0.62	0.55	0.73
ZvZ	0.69	0.64	0.67	0.73	0.74	0.77	0.7	0.73	0.73	0.83	0.85	0.85	0.81	0.89	0.94	0.81	0.88
overall	0.68	0.62	0.68	0.73	0.76	0.78	0.69	0.76	0.77	0.63	0.67	0.67	0.63	0.75	0.75	0.63	0.75

be serialized and done only once when the dataset changes. The prediction computation corresponds to the sum in the question (III.B.5) and so its computational complexity is in $O(N \cdot M)$ with N build trees and M possible observations, as $M << N$, we can consider it linear in the number of build trees (values of $BuildTree$).

Table 7.4: Extremes of computation time values (in seconds, C2D 2.8Ghz)

Race	Nb Games	Learning time	Inference μ	Inference σ^2
T (max)	1036	0.197844	0.0360234	0.00892601
T (Terran)	567	0.110019	0.030129	0.00738386
P (Protoss)	1021	0.13513	0.0164457	0.00370478
P (Protoss)	542	0.056275	0.00940027	0.00188217
Z (Zerg)	1028	0.143851	0.0150968	0.00334057
Z (Zerg)	896	0.089014	0.00796715	0.00123551

Extensions

Possible Uses

Developing beforehand a RTS game AI that specifically deals with whatever strategies the players will come up is very hard. And even if game developers were willing to patch their AI afterwards, it would require a really modular design and a lot of work to treat each strategy. With our model, the AI can adapt to the evolutions in play by learning its parameters from the replay, and it can dynamically adapt during the games by using the reverse question $P(BuildTree|Opening, Time)$, or even $P(TechTree|Opening, Time)$ if we use a $TechTree$ variable encoding buildings and technology upgrades. This question would give the distribution over technology trees knowing the opening we want to perform at which time. This would allow for the bot to dynamically choose/change build orders.

We could also use this model in a commentary assistant AI. In the StarCraft and StarCraft 2 communities, there are a lot of pro-gamers* tournaments that are commented and we could provide a tool for commentators to estimate the probabilities or different openings or technology paths. As in commented poker matches, where the probabilities of different hands are drawn on screen for the spectators, we could display the probabilities of openings. In such a setup we could use more features as the observers and commentators can see everything that happens (upgrades, units) and we limited ourselves to "key" buildings in the work presented in this paper.

Improvements

First, our prediction model can be upgraded to have a higher recognition rate: we could reason about $t+1$ explicitly before computing the distribution over possible openings at t and thus compute the distribution over technology trees at $t+1$. Perhaps it would increase the results of $P(Opening|Observations)$, but it almost surely would increase $P(BuildTree^{t+1}|Observations)$ which is important for late game predictions. We could also make use of more features as we currently only use at most 20 features (only buildings), and never all at once. Perhaps also that incorporating priors per match-up would lead to better results.

Then, we could feed it with *more* replays during the learning by scrapping more progamers level replays websites. Also, we could learn from replays of bot vs bot matches. For the learning part, the labeling of replays is very important, and our labeling methods can be improved. We could explore auto-supervised learning [**?**]. Clearly, some match-ups are handled better, either in the replays labeling part and/or in the prediction part. Replays could be labeled by humans and we would do supervised learning then. Or they could be labeled by a combination of rules (as in [**?**]) and statistical analysis (as the method presented here). Finally, the replays could be labeled by match-up dependent openings (instead of race dependent openings currently) and could contain either the two parts of the opening or the game time at which the label is the most relevant, as openings are often bimodal ("fast expand into mutas", "corsairs into reaver", etc.).

Finally, a hard problem is detecting the "fake" builds of very highly skilled players. Indeed, some progamers have build orders which purpose are to fool the opponent into thinking that they are performing opening A while they are doing B. For instance, they could leadthe opponent to think they are going to *tech* and perform an early rush instead. We think that this can be handled by our model by changing $P(Opening|LastOpening)$ by $P(Opening|LastOpening, LastObservations)$ and adapting the influence of the last prediction with regard to the last observations (i.e., we think we can learn some "fake" label on replays).

7.3 Strategy adaptation

Related Works

Strategy Adaptation Model

Results on StarCraft

Extensions

Chapter 8

Inter-game Adaptation ("meta-game")

Other cases of learning not dealt with in previous chapters:

8.1 Player modeling

$$P(WhatToDo|Opponent)$$

Basic adaptation to opponents' play styles between games.

8.2 Reinforcement learning

$$P(WhatToDo|Ourself)$$

Seeking the causes of success/failures and modifying parameters accordingly. The farther from the evaluated model you are, the hardest is the modification/search/tuning.

8.3 Discussion

$$P(WhatToDo|IThink').P(IThink'|HeThinks).P(HeThinks|IThink)...$$

Meta-game / Psychological warfare / Game theory / I think he thinks I think...

Chapter 9

BroodwarBotQ: putting it all together

9.1 Code Architecture

mapping schéma code <-> schéma info-flow.

9.2 A Game Walkthrough

The tree of decisions.

9.3 Results

AIIDE 2010,2011, Ladder.

Chapter 10

Conclusion

10.1 Contrib

Résumer les contributions

Approaches

- train your IA from data
- or train your IA by itself
- or let the game designers set the parameters

Results

Recall what works, what should be extended.

10.2 Perspectives: Not a solved problem yet

Computers don't beat good (experts) humans (higher level strategic thinking: common sense, plus vision/interpolation for efficient micro). They are not so fun (do not adapt that much, our bot is the most adaptive ATM). Competition results. Tout ce qui peut se faire en recherche et ce qui est directement applicable par l'industrie.

Glossary

AI directors system that overlooks the behavior of the players to manage the intensity, difficulty and fun. 8, 70

CBR case-based reasoning. 22, 70

Dark Spore a fast-paced, sci-fi action-RPG, with PvP and cooperative (vs AI) modes. 8, 70

DSL Domain Specific Language. 9, 70

FPS First Person Shooter: egocentric shooter game, strong sub-genres are fast FPS, also called "Quake-likes", e.g. Quake III; and team/tactical FPS, e.g. Counter-Strike, Team Fortress 2. 8, 70

gameplay describes the interactive aspects of game design, which constraints the players possible behaviors and the players' goals in a given game. 18, 47, 70

goban board used for the game of Go. 14, 70

Left 4 Dead a teamplay (cooperative vs AI) survival horror FPS in which players have to fight and escape zombie hordes. 8, 70

MCTS Monte-Carlo Tree Search. 14, 70

MMORPG Massively Multi-player Online Role Playing Game, distinct of RPG by the scale of cooperation sometimes needed to achieve a common goal, e.g. Dark Age of Camelot, World of Warcraft. 8, 70

NPC non-playing characters. 7, 8, 70

opening in Chess as in RTS games: the first strategic moves of the game, the strategy of the early game. 52, 70

partisan (game) which is not impartial, in which a player can do actions another can not do (move a faction while the other player(s) cannot). 10, 70

perfect-information (game) in which all the players have complete knowledge of the (board) state of the game. 10, 70

positional hashing a method for determining similarities in (board) positions using hash functions. 15, 70

pro-gamer professional gamer, full-time job. 7, 63, 70

PvE Players vs Environment. 8, 70

PvP Players versus Players. 8, 70

replay the record of all players' actions during a game, allowing the game engine to recreate the game state deterministically. 8, 70

RPG Role Playing Game, e.g. Dungeons & Dragons based Baldur's Gate. 8, 70

RTS Real-Time Strategy games are (mainly) allocentric economic and military simulations from an operational tactical/strategist commander viewpoint, e.g. Command & Conquer, Age of Empires, StarCraft, Total Annihilation. 8, 70

solved game a game whose outcome can be correctly predicted from any position when each side plays optimally. 10, 70

StarCraft: Brood War a science fiction real-time strategy (RTS) game released in 1998 by Blizzard Entertainment. 7, 70

UCT Upper Confidence Bounds for Trees. 14, 70

zero-sum game a game in which the total score of each players, from one player's point-of-view, for every possible strategies, adds up to zero; *i.e.* "a player benefits only at the expense of others". 10, 70

Bibliography

David W. Aha, Matthew Molineaux, and Marc J. V. Ponsen. Learning to win: Case-based plan selection in a real-time strategy game. In Héctor Muñoz-Avila and Francesco Ricci, editors, ICCBR, volume 3620 of Lecture Notes in Computer Science, pages 5–20. Springer, 2005. ISBN 3-540-28174-6.

Victor L. Allis. Searching for Solutions in Games and Artificial Intelligence. PhD thesis, University of Limburg, 1994. URL http://fragrieu.free.fr/SearchingForSolutions.pdf.

Robert B. Ash and Richard L. Bishop. Monopoly as a Markov process. Mathematics Magazine, (45): 26–29, 1972.

John Asmuth, Lihong Li, Michael Littman, Ali Nouri, and David Wingate. A bayesian sampling approach to exploration in reinforcement learning. In Uncertainty in Artificial Intelligence, UAI, pages 19–26. AUAI Press, 2009.

Radha-Krishna Balla and Alan Fern. Uct for tactical assault planning in real-time strategy games. In International Joint Conference of Artificial Intelligence, IJCAI, pages 40–45, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.

Matthew J. Beal. Variational algorithms for approximate bayesian inference. PhD. Thesis, 2003.

Curt Bererton. State estimation for game ai using particle filters. In AAAI Workshop on Challenges in Game AI, 2004.

Pierre Bessière, Christian Laugier, and Roland Siegwart. Probabilistic Reasoning and Decision Making in Sensory-Motor Systems. Springer Publishing Company, Incorporated, 1st edition, 2008. ISBN 3540790063, 9783540790068.

Darse Billings, Jonathan Schaeffer, and Duane Szafron. Poker as a testbed for machine intelligence research. In Advances in Artificial Intelligence, pages 1–15. Springer Verlag, 1998.

Darse Billings, Neil Burch, Aaron Davidson, Robert C. Holte, Jonathan Schaeffer, Terence Schauenberg, and Duane Szafron. Approximating game-theoretic optimal strategies for full-scale poker. In Georg Gottlob and Toby Walsh, editors, IJCAI, pages 661–668. Morgan Kaufmann, 2003.

Michael Booth. The AI Systems of Left 4 Dead. In Proceedings of the Fifth Artificial Intelligence and Interactive Digital E The AAAI Press, 2009. URL http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf.

C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. Computational Intelligence and AI in Games, IEEE Transactions on, PP(99):1, 2012. ISSN 1943-068X. doi: 10.1109/TCIAIG.2012.2186810.

Michael Buro. Call for ai research in rts games. In In Proceedings of the AAAI Workshop on AI in Games, pages 139–141. AAAI Press, 2004.

Simon Butler and Yiannis Demiris. Partial observability during predictions of the opponent's movements in an rts game. In CIG (IEEE), 2010.

Pedro Cadena and Leonardo Garrido. Fuzzy case-based reasoning for managing strategic and tactical reasoning in starcraft. In MICAI (1), pages 113–124, 2011.

Murray Campbell, A. Joseph Hoane Jr., and Feng hsiung Hsu. Deep blue. Artif. Intell., 134(1-2):57–83, 2002.

Alex Champandard, Tim Verweij, and Remco Straatman. Killzone 2 multiplayer bots. In Paris Game AI Conference, 2009.

Michael Chung, Michael Buro, and Jonathan Schaeffer. Monte carlo planning in rts games. In CIG. IEEE, 2005.

David Churchill and Michael Buro. Build order optimization in starcraft. In Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2011.

CitationNeeded. needed, 0000.

Francis Colas, Julien Diard, and Pierre Bessière. Common bayesian models for common cognitive issues. Acta Biotheoretica, 58:191–216, 2010. ISSN 0001-5342.

Contracts. Progamers income list: `http://www.teamliquid.net/forum/viewmessage.php?topic_id=49725`, 2007.

Maria Cutumisu and Duane Szafron. An architecture for game behavior ai: Behavior multi-queues. In AAAI, editor, AIIDE, 2009.

Holger Danielsiek, Raphael Stür, Andreas Thom, Nicola Beume, Boris Naujoks, and Mike Preuss. Intelligent moving of groups in real-time strategy games. In Philip Hingston and Luigi Barone, editors, CIG, pages 71–78. IEEE, 2008. ISBN 978-1-4244-2973-8.

Ethan Dereszynski, Jesse Hostetler, Alan Fern, Tom Dietterich Thao-Trang Hoang, and Mark Udarbe. Learning probabilistic behavior models in real-time strategy games. In AAAI, editor, Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2011.

Julien Diard, Pierre Bessière, and Emmanuel 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.

Kutluhan Erol, James Hendler, and Dana S. Nau. Htn planning: Complexity and expressivity. In In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94, pages 1123–1128. AAAI Press, 1994.

Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. Artificial Intelligence, 2(3–4):189 – 208, 1971. ISSN 0004-3702. doi: 10.1016/0004-3702(71)90010-5. URL `http://www.sciencedirect.com/science/article/pii/0004370271900105`.

Kenneth D. Forbus, James V. Mahoney, and Kevin Dill. How qualitative spatial reasoning can improve strategy game ais. IEEE Intelligent Systems, 17:25–30, July 2002. ISSN 1541-1672. doi: http://dx.doi.org/10.1109/MIS.2002.1024748. URL `http://dx.doi.org/10.1109/MIS.2002.1024748`.

Colin Frayn. An evolutionary approach to strategies for the game of monopoly. In CIG, 2005.

Sylvain Gelly and Yizao Wang. Exploration exploitation in Go: UCT for Monte-Carlo Go. In NIPS: Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop Canada, December 2006. URL http://hal.archives-ouvertes.fr/hal-00115330/en/.

Sylvain Gelly, Yizao Wang, Rémi Munos, and Olivier Teytaud. Modification of UCT with Patterns in Monte-Carlo Go. Rapport de recherche RR-6062, INRIA, 2006. URL http://hal.inria.fr/inria-00117266.

E. A. A. Gunn, B. G. W. Craenen, and E. Hart. A Taxonomy of Video Games and AI. In AISB 2009, April 2009.

Johan Hagelbäck and Stefan J. Johansson. A multiagent potential field-based bot for real-time strategy games. Int. J. Comput. Games Technol., 2009:4:1–4:10, January 2009. ISSN 1687-7047. doi: http://dx.doi.org/10.1155/2009/910819. URL http://dx.doi.org/10.1155/2009/910819.

Johan Hagelbäck and Stefan J. Johansson. A study on human like characteristics in real time strategy games. In CIG (IEEE), 2010.

Robert A. Hearn and Erik D. Demaine. Games, Puzzles, and Computation. A K Peters, July 2009.

P Hingston. A turing test for computer game bots. IEEE Transactions on Computational Intelligence and AI in Games, 1(3):169–186, 2009. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5247069.

Stephen Hladky and Vadim Bulitko. An evaluation of models for predicting opponent positions in first-person shooter video games. In CIG (IEEE), 2008.

Ryan Houlette and Dan Fu. The ultimate guide to fsms in games. AI Game Programming Wisdom 2, 2003.

Ji-Lung Hsieh and Chuen-Tsai Sun. Building a player strategy model by analyzing replays of real-time strategy games. In IJCNN, pages 3106–3111. IEEE, 2008.

Damian Isla. Handling complexity in the halo 2 ai. In Game Developers Conference, 2005.

Froduald Kabanza, Philipe Bellefeuille, Francis Bisson, Abder Rezak Benaskeur, and Hengameh Irandoust. Opponent behaviour recognition for real-time strategy games. In AAAI Workshops, 2010. URL http://aaai.org/ocs/index.php/WS/AAAIW10/paper/view/2024.

Jaekwang Kim, Kwang Ho Yoon, Taebok Yoon, and Jee-Hyong Lee. Cooperative learning by replay files in real-time strategy game. In Proceedings of the 7th international conference on Cooperative design, visualization, and engineering, CDVE'10, pages 47–51, Berlin, Heidelberg, 2010. Springer-Verlag. ISBN 3-642-16065-4, 978-3-642-16065-3. URL http://portal.acm.org/citation.cfm?id=1887315.1887323.

Dan Kline. Bringing Interactive Storytelling to Industry. In Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2009). The AAAI Press, 2009. URL http://dankline.files.wordpress.com/2009/10/bringing-interactive-story-to-industry-aiide-09.ppt.

Dan Kline. The ai director in dark spore. In Paris Game AI Conference, 2011. URL http://dankline.files.wordpress.com/2011/06/ai-director-in-darkspore-gameai-2011.pdf.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In In: ECML-06. Number 4212 in LNCS, pages 282–293. Springer, 2006.

Daphne Koller and Avi Pfeffer. Representations and solutions for game-theoretic problems. Artificial Intelligence, 94:167–215, 1997.

Kevin B. Korb, Ann E. Nicholson, and Nathalie Jitnah. Bayesian poker. In In Uncertainty in Artificial Intelligence, pages 343–350. Morgan Kaufman, 1999.

John E. Laird. It knows what you're going to do: adding anticipation to a quakebot. In Agents, pages 385–392, 2001.

John E. Laird and Michael van Lent. Human-level ai's killer application: Interactive computer games. AI Magazine, 22(2):15–26, 2001.

Ronan Le Hy, Anthony Arrigoni, Pierre Bessière, and Olivier Lebeltel. Teaching bayesian behaviours to video game characters, 2004. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.3935.

Olivier Lebeltel, Pierre Bessière, Julien Diard, and Emmanuel Mazer. Bayesian robot programming. Autonomous Robots, 16(1):49–79, 2004. ISSN 0929-5593.

Daniele Loiacono, Julian Togelius, Pier Luca Lanzi, Leonard Kinnaird-Heether, Simon M. Lucas, Matt Simmerson, Diego Perez, Robert G. Reynolds, and Yago Sáez. The wcci 2008 simulated car racing competition. In CIG, pages 119–126, 2008.

I. Lynce and J. Ouaknine. Sudoku as a sat problem. Proc. of the Ninth International Symposium on Artificial Intelligence and Mathematics. Springer, 2006.

David J. C. MacKay. Information Theory, Inference, and Learning Algorithms. Cambridge University Press, 2003.

Charles Madeira, Vincent Corruble, and Geber Ramalho. Designing a reinforcement learning-based adaptive AI for large-scale strategy games. In AI and Interactive Digital Entertainment Conference, AIIDE (AAAI), 2006.

Ashwin Ram Manish Meta, Santi Ontanon. Meta-level behavior adaptation in real-time strategy games. In ICCBR-10 Workshop on Case-Based Reasoning for Computer Games, Alessandria, Italy, 2010.

Bhaskara Marthi, Stuart Russell, David Latham, and Carlos Guestrin. Concurrent hierarchical reinforcement learning. In IJCAI, pages 779–785, 2005.

Chris Miles, Juan C. Quiroz, Ryan E. Leigh, and Sushil J. Louis. Co-evolving influence map tree based strategy game players. In CIG, pages 88–95. IEEE, 2007. ISBN 1-4244-0709-5.

Kinshuk Mishra, Santiago Ontañón, and Ashwin Ram. Situation assessment for plan retrieval in real-time strategy games. In Klaus-Dieter Althoff, Ralph Bergmann, Mirjam Minor, and Alexandre Hanft, editors, ECCBR, volume 5239 of Lecture Notes in Computer Science, pages 355–369. Springer, 2008. ISBN 978-3-540-85501-9.

Matthew Molineaux, David W. Aha, and Philip Moore. Learning continuous action models in a real-time strategy strategy environment. In FLAIRS Conference, pages 257–262, 2008.

John Nash. Non-cooperative games. The Annals of Mathematics, 54(2):286–295, 1951. URL http://jmvidal.cse.sc.edu/library/nash51a.pdf.

Santiago Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. Case-based planning and execution for real-time strategy games. In Proceedings of the 7th international conference on Case-Based Reasoning: Case-Based Reasoning Research and Development, ICCBR '07, pages 164–178, Berlin, Heidelberg, 2007. Springer-Verlag. ISBN 978-3-540-74138-1. doi: http://dx.doi.org/10.1007/978-3-540-74141-1_12. URL http://dx.doi.org/10.1007/978-3-540-74141-1_12.

Santi Ontañón, Kinshuk Mishra, Neha Sugandh, and Ashwin Ram. On-line case-based planning. Computational Intelligence, 26(1):84–119, 2010.

Jeff Orkin. Three states and a plan: The a.i. of f.e.a.r. In GDC, 2006.

Martin J. Osborne and Ariel Rubinstein. A course in game theory. The MIT Press, July 1994. ISBN 0262650401. URL http://www.worldcat.org/isbn/0262650401.

Luke Perkins. Terrain analysis in real-time strategy games: An integrated approach to choke point detection and region decomposition. In G. Michael Youngblood and Vadim Bulitko, editors, AIIDE. The AAAI Press, 2010.

Aske Plaat. Research, re: search and re-search. PhD thesis, Erasmus University Rotterdam, 1996.

Marc Ponsen and Ir. P. H. M. Spronck. Improving Adaptive Game AI with Evolutionary Learning. PhD thesis, University of Wolverhampton, 2004.

Mike Preuss, Nicola Beume, Holger Danielsiek, Tobias Hein, Boris Naujoks, Nico Piatkowski, Raphael Stüer, Andreas Thom, and Simon Wessing. Towards intelligent team composition and maneuvering in real-time strategy games. Transactions on Computational Intelligence and AI in Games, 2(2):82–98, June 2010.

Steve Rabin. Implementing a state machine language. AI Game Programming Wisdom, pages 314—-320, 2002.

Miquel Ramírez and Hector Geffner. Plan recognition as planning. In Proceedings of the 21st international joint conference on Artifical intelligence, pages 1778–1783. Morgan Kaufmann Publishers Inc., 2009.

Alexander Reinefeld. An Improvement to the Scout Tree-Search Algorithm. International Computer Chess Association Journal, 6(4):4–14, December 1983.

Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In Proceedings of the 14th annual conference on Computer graphics and interactive techniques, SIGGRAPH '87, pages 25–34, New York, NY, USA, 1987. ACM. ISBN 0-89791-227-6. doi: 10.1145/37401.37406. URL http://doi.acm.org/10.1145/37401.37406.

Mark Riedl, Boyang Li, Hua Ai, and Ashwin Ram. Robust and authorable multiplayer storytelling experiences. 2011. URL http://www.aaai.org/ocs/index.php/AIIDE/AIIDE11/paper/view/4068/4434.

Philipp Rohlfshagen and Simon M. Lucas. Ms pac-man versus ghost team cec 2011 competition. In IEEE Congress on Evolutionary Computation, pages 70–77, 2011.

F. Schadd, S. Bakkes, and P. Spronck. Opponent modeling in real-time strategy games. pages 61–68, 2007.

Jonathan Schaeffer, Yngvi Björnsson Neil Burch, Akihiro Kishimoto, Martin Müller, Rob Lake, Paul Lu, and Steve Sutphen. Checkers is solved. Science, 317(5844):1518–1522, 2007. Work named by Science Magazine as one of the 10 most important scientific achievements of 2007.

Jacob Schrum, Igor V. Karpov, and Risto Miikkulainen. Ut2: Human-like behavior via neuroevolution of combat behavior and replay of human traces. In Proceedings of the IEEE Conference on Computational Intelligence and Games (CIG 2011), pages 329–336, Seoul, South Korea, September 2011. IEEE. URL http://nn.cs.utexas.edu/?schrum:cig11competition.

N. Shaker, J. Togelius, and G. N. Yannakakis. Towards Automatic Personalized Content Generation for Platform Games. In Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE). AAAI Press, October 2010.

C E Shannon. Programmig a computer for chess playing. Philopophical Magazine, 1950.

Manu Sharma, Michael Holmes, Juan Santamaria, Arya Irani, Charles L. Isbell, and Ashwin Ram. Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. In International Joint Conference of Artificial Intelligence, IJCAI, 2007.

Helmut Simonis. Sudoku as a constraint problem. CP Workshop on modeling and reformulating Constraint Satisfaction Problems, page 13–27, 2005. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.2964&rep=rep1&type=pdf.

Greg Smith, Phillipa Avery, Ramona Houmanfar, and Sushil Louis. Using co-evolved rts opponents to teach spatial tactics. In CIG (IEEE), 2010.

Finnegan Southey, Michael Bowling, Bryce Larson, Carmelo Piccione, Neil Burch, Darse Billings, and Chris Rayner. Bayes' bluff: Opponent modelling in poker. In In Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI, pages 550–558, 2005.

Finnegan Southey, Wesley Loh, and Dana Wilkinson. Inferring complex agent motions from partial trajectory observations. In Proceedings of the 20th international joint conference on Artifical intelligence, pages 2631–2637, San Francisco, CA, USA, 2007. Morgan Kaufmann Publishers Inc. URL http://portal.acm.org/citation.cfm?id=1625275.1625699.

Richard S. Sutton and Andrew G. Barto. Reinforcement Learning: An Introduction (Adaptive Computation and Machine The MIT Press, March 1998. ISBN 0262193981.

Gabriel Synnaeve and Pierre Bessière. A Bayesian Model for Plan Recognition in RTS Games applied to StarCraft. In AAAI, editor, Proceedings of the Seventh Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2011), Proceedings of AIIDE, pages 79–84, Palo Alto, États-Unis, October 2011. URL http://hal.archives-ouvertes.fr/hal-00641323/en/. 7 pages.

Gabriel Synnaeve and Pierre Bessière. Bayesian Modeling of a Human MMORPG Player. In 30th international workshop on Bayesian Inference and Maximum Entropy, Chamonix, France, July 2010. URL http://hal.inria.fr/inria-00538744.

Gabriel Synnaeve and Pierre Bessière. A Bayesian Model for RTS Units Control applied to StarCraft. In Proceedings of IEEE CIG 2011, page 000, Seoul, Corée, République De, September 2011a. URL http://hal.archives-ouvertes.fr/hal-00607281/en/.

Gabriel Synnaeve and Pierre Bessière. A Bayesian Model for Opening Prediction in RTS Games with Application to StarCraft. In Proceedings of 2011 IEEE CIG, page 000, Seoul, Corée, République De, September 2011b. URL http://hal.archives-ouvertes.fr/hal-00607277/en/.

Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. The 2009 mario ai competition. In IEEE Congress on Evolutionary Computation, pages 1–8, 2010.

John Tromp and Gunnar Farnebäck. Combinatorics of Go. Submitted to CG 2006, 2006. URL http://homepages.cwi.nl/~{}tromp/go/gostate.ps.

Andrew Trusty, Santiago Ontañón, and Ashwin Ram. Stochastic plan optimization in real-time strategy games. In Christian Darken and Michael Mateas, editors, AIIDE. The AAAI Press, 2008. ISBN 978-1-57735-391-1.

William van der Sterren. Multi-unit planning with htn and a*. In Paris Game AI Conference, 2009.

J.M.P. van Waveren and L.J.M. Rothkrantz. Artificial player for quake iii arena. International Journal of Intelligent Games & Simulation (IJIGS), 1(1):25–32, March 2002.

John Von Neumann and Oskar Morgenstern. Theory of Games and Economic Behavior. Princeton University Press, 1944. URL http://jmvidal.cse.sc.edu/library/neumann44a.pdf.

Ben G. Weber and Michael Mateas. A data mining approach to strategy prediction. In CIG (IEEE), 2009.

Ben G. Weber and Santiago Ontañón. Using automated replay annotation for case-based planning in games. In ICCBR Workshop on CBR for Computer Games (ICCBR-Games), 2010.

Ben G. Weber, Michael Mateas, and Arnav Jhala. Applying goal-driven autonomy to starcraft. In Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2010a.

Ben G. Weber, Peter Mawhorter, Michael Mateas, and Arnav Jhala. Reactive planning idioms for multi-scale game ai. In CIG (IEEE), 2010b.

Ben G. Weber, Michael Mateas, and Arnav Jhala. A particle model for state estimation in real-time strategy games. In Proceedings of AIIDE, page 103–108, Stanford, Palo Alto, California, 2011. AAAI Press, AAAI Press.

Joost Westra and Frank Dignum. Evolutionary neural networks for non-player characters in quake iii. In CIG (IEEE), 2009.

Samuel Wintermute, Joseph Z. Joseph Xu, and John E. Laird. Sorts: A human-level approach to real-time strategy ai. In AIIDE, pages 55–60, 2007.

Stephano Zanetti and Abdennour El Rhalibi. Machine learning techniques for fps in q3. In Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology, ACE '04, pages 239–244, New York, NY, USA, 2004. ACM. ISBN 1-58113-882-2. doi: http://doi.acm.org/10.1145/1067343.1067374. URL http://doi.acm.org/10.1145/1067343.1067374.

Appendix A

Game AI

Algorithm 5 Negamax algorithm

```
function NEGAMAX(depth)
    if depth ≤ 0 then
        return value()
    end if
    α = −∞
    for all possible moves do
        α = max(α, −negamax(depth − 1))
    end for
    return α
end function
```