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 progamers was StarCraft: Brood War, in Korea, with top players earning more than 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, re) 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: game directors (Left 4 Dead, Dark Spore), procedural content generation (Mario PCG).

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:

- computational efficiency (particularly for real-time games with advanced graphics, the AI CPU budget is low),

[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.

- game designers often want to remain in control of the behaviors (and editing tools have to be usable by them),

- scalability (as much autonomy as possible), "debugability" (huge states spaces due to the presence of the player(s)), 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 (ain't that 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.

2.2 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.

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

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 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

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
```

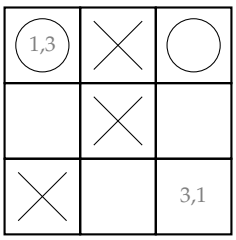

Figure 2.1: A Tic-tac-toe board position, "circles" turn to play. The couples of numbers explain the numbering (left to right, bottom to top, starting at 1) of the grid.

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 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

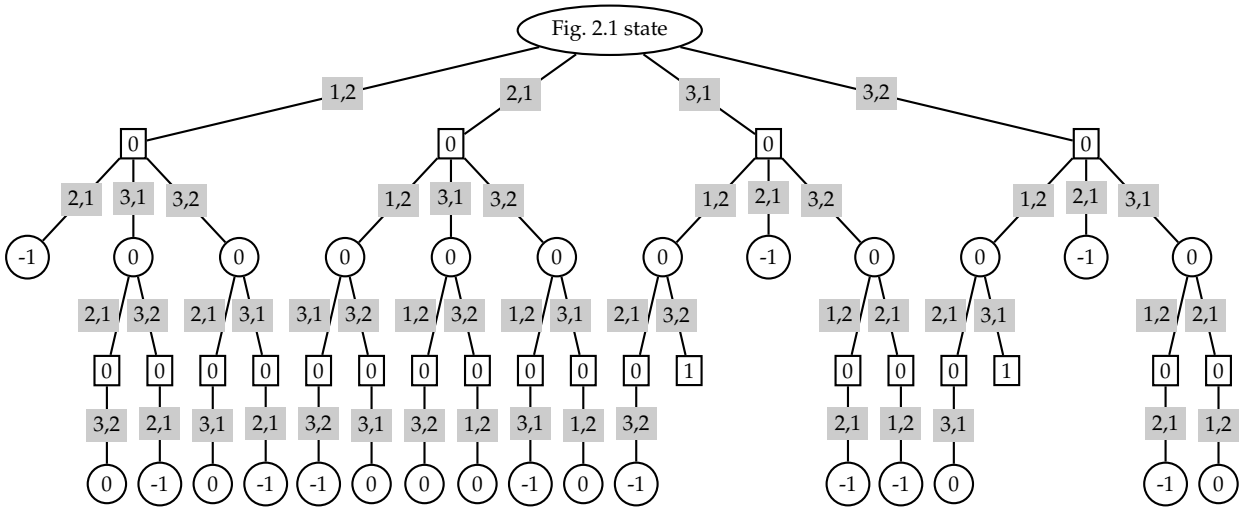

Figure 2.2: Minimax tree with initial position at Fig. 2.1 state, **nodes** are states and **edges** are transitions, labeled with the move. Leafs are end-game states: 1 point for the win and -1 for the loss. Player is "circles" and plays first (first edges are player's moves).

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
```

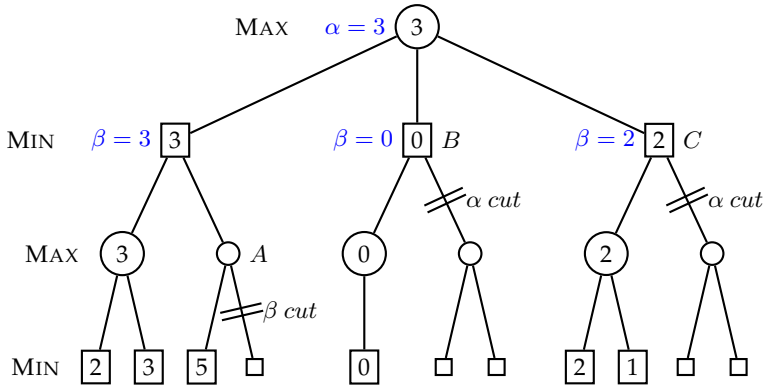

Figure 2.3: Alpha-beta cuts on a Minimax tree, **nodes** are states and **edges** are transitions, labeled with the values of positions at the bottom (max depth). Here is the trace of the algorithm: **1.** descend leftmost first and evaluated 2 and 3, **2.** percolate max(2,3) higher up to set $\beta = 3$, **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.

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 or MTD-f), 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 avantageous 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**
    $tree \leftarrow$ EXPANDFROM($node$)
    $tree.values \leftarrow$ 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 $\exists c \in node.children \; s.t. \; N(c) = 0$ **then**
    return c `// grow`
  end if
  return EXPANDFROM($\arg\max_{c \in node.children)} \frac{Q(c)}{N(c)} + k\sqrt{\frac{\ln N(node)}{N(c)}}$) `// 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 a multi-armed bandit problem and so EXPANDFROM($node$) UCB1 chooses the arms with the best upper confidence bound:

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

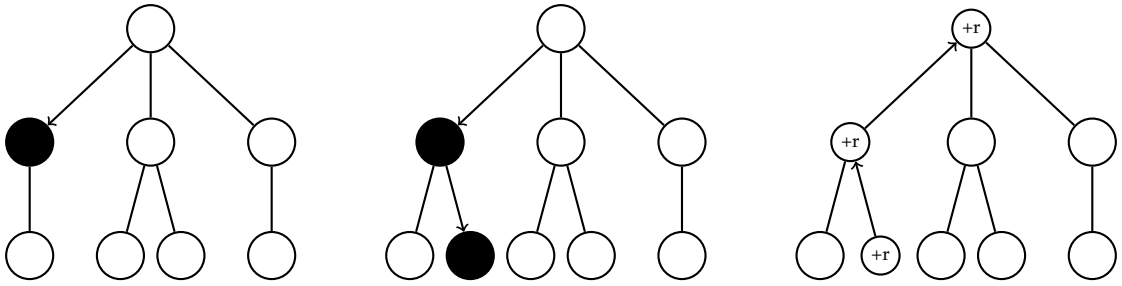

Figure 2.4: An iteration of the **while** loop in MCTS, from left to right: EXPANDFROM select, EXPANDFROM grow, EVALUATE & MERGE.

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

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

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 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.

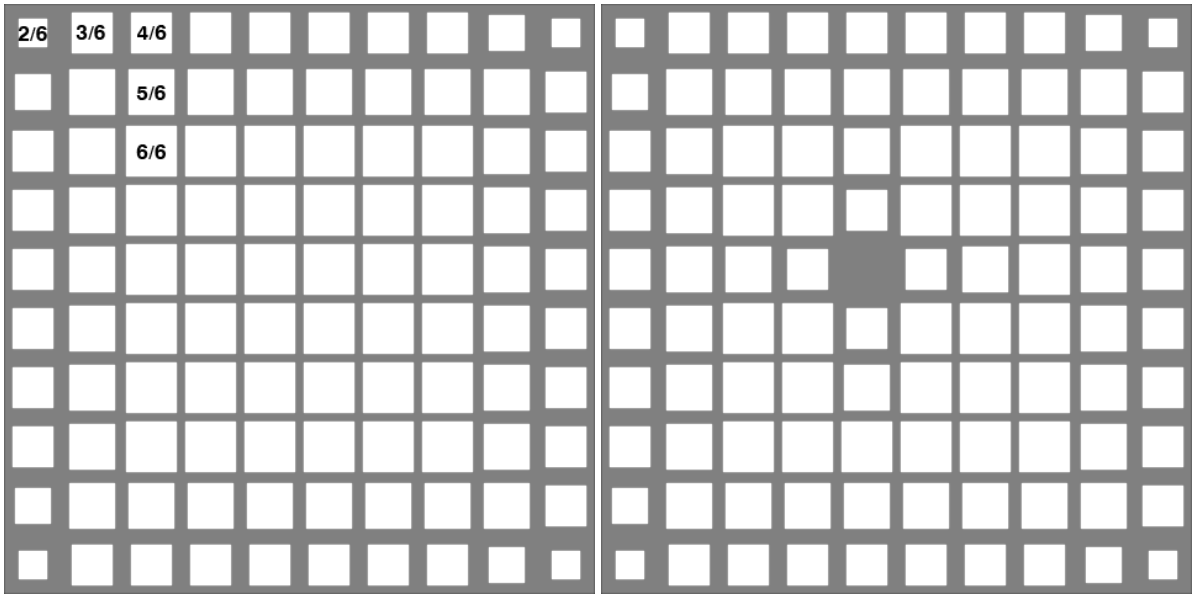

Figure 2.5: Left: visualization of probabilities for squares to contain a ship of size 3 ($P(O_{i,j}) = true$) initially, assuming uniform distribution of this type of ship. Annotations correspond to the *number of combinations* (on six, the maximum number of conformations), Right: same probability after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

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

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), 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, teamplay 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://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. krishna 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.

Challenges

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.

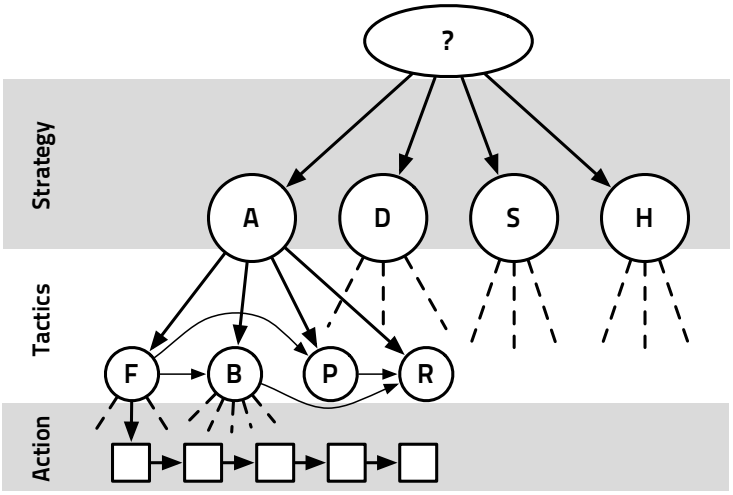

Figure 2.6: Abstract decision hierarchy in a video game

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

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

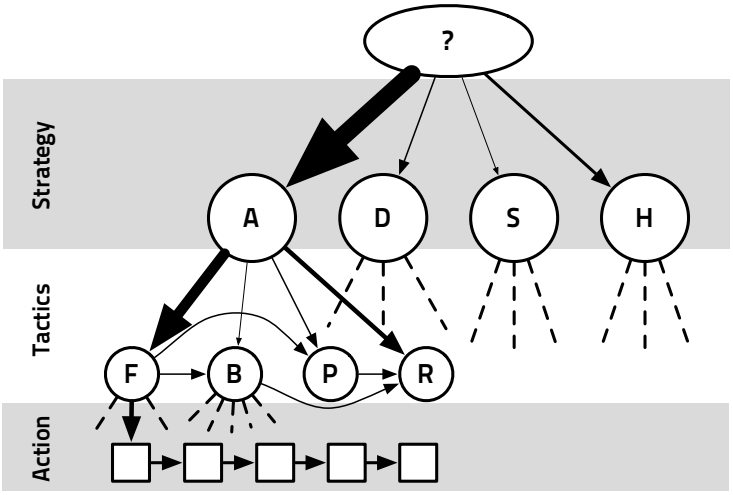

Figure 2.7: Vertical continuity in decision-making in a video game

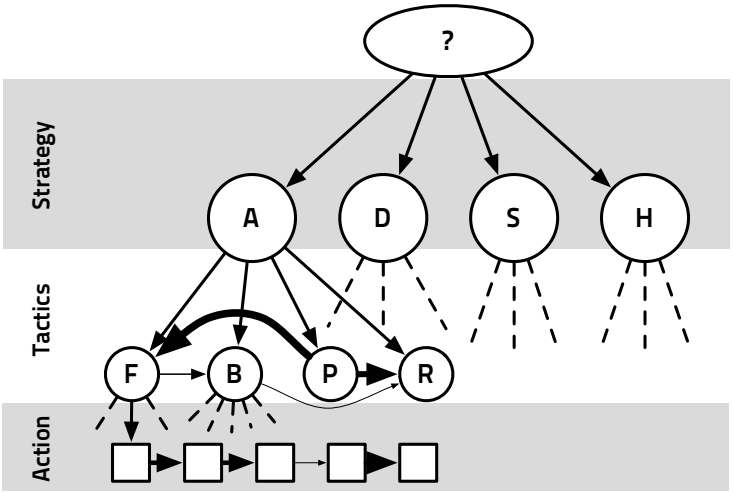

Figure 2.8: Horizontal continuity in decision-making in a video game

Vertical continuity

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

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.

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	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 progamers, counting group actions as only one action

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

Our thesis: 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

Player modeling & plan recognition

Strategy adaptation

Operational team decision making

Micro / Tactics (squad AI)

Planning under uncertainty

Macro

Learning and adaptability

Multi-scale

3.2 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-)

3.3 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

Chapter 4

Bayesian programming and modeling

4.1 Bayesian programming

4.2 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

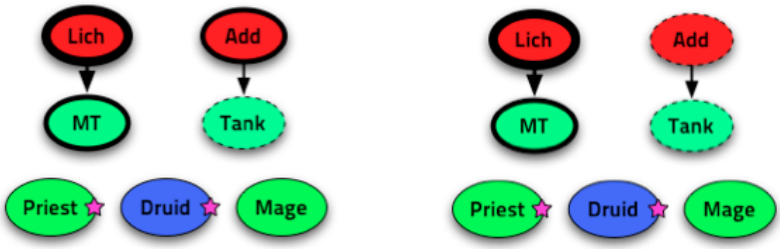

Figure 4.1: Example setup A (left) and B (right), 2 foes, 2 "tanks", players with stars can heal allies, players with dotted lines will soon die ($ID = true$).

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. 4.1 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. 4.2 (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. 4.2 (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.

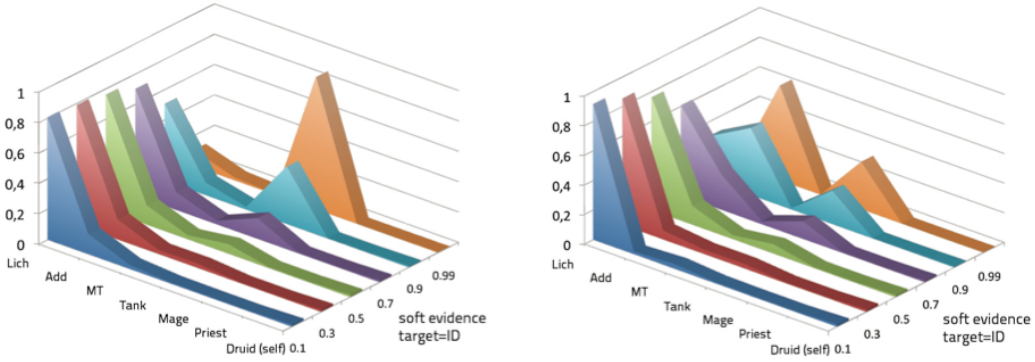

Figure 4.2: Left: probabilities of targets depending on the soft evidence that a target is dying with setup A. Right: same, with setup B.

For the "skill selection" model, we can see on Fig. 4.3 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. 4.4 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.

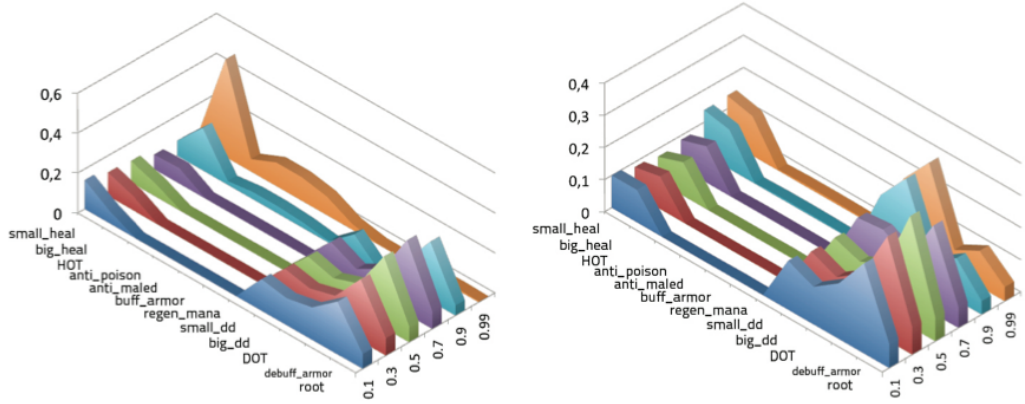

Figure 4.3: Left: Probabilities of skills depending on the soft evidence that a target is dying with setup A. Right: same, with setup B.

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

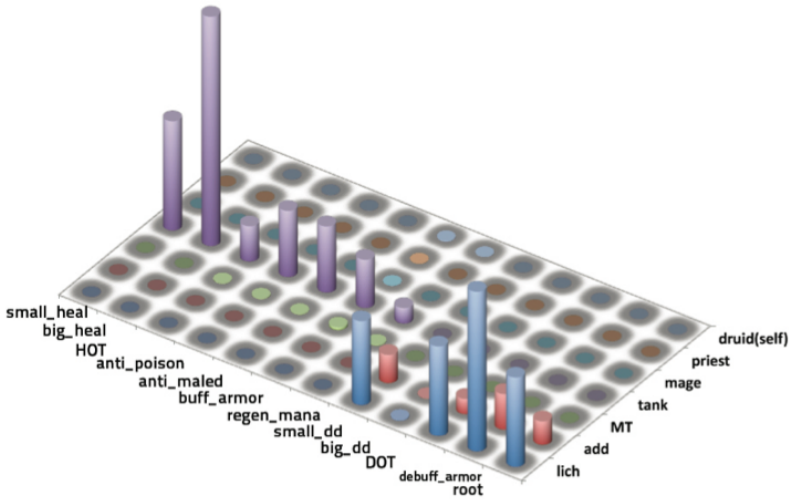

Figure 4.4: Log-probabilities of Target and Skill with setup A, and $P(ID|Target) = 0.9, P(A|Target) = 0.6$

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:

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

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

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

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

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

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 5

RTS AI: *StarCraft: Broodwar*

5.1 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.

5.2 Problems, resolutions

5.3 Task decomposition and linking

- 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

Chapter 6

Micro

- Problem: (optimal) control of units in a (real-continuous-time) huge actions space
- Complexity: P-space
- State of the art: de-centralized rules, [Marthi et al., 2005], [Weber et al., 2010b]
- Our take: [Synnaeve and Bessière, 2011a]
- Results: more or less state of the art
- Conclusion and perspectives: opens the game to parameters learning, RL and/or EA (Bayesian fusion FTW!)

Chapter 7

Tactics

- 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, krishna 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).

Chapter 8

Strategy

- 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 $\Rightarrow$ (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

Chapter 9

Meta-Reasoning

learning to learning applied to learning

9.1 Player modeling

Basic adaptation to opponents play between games and during a game.

9.2 Reinforcement learning

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.

9.3 Meta-game / Psychological warfare / Game theory

I think he thinks I think...

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

Racall 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

DSL Domain Specific Language. 50

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. 50

MCTS Monte-Carlo Tree Search. 50

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. 50

partisan game a 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). 50

perfect-information game a game in which all the players have complete knowledge of the (board) state of the game. 50

PvE Players vs Environment. 50

PvP Players versus Players. 50

RPG Role Playing Game, e.g. Dungeons & Dragons based Baldur's Gate. 50

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. 50

solved game a game whose outcome can be correctly predicted from any position when each side plays optimally. 50

UCT Upper Confidence Bounds for Trees. 50

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". 50

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.

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

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.

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.

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.

Radha krishna Balla and Alan Fern. Uct for tactical assault planning in real-time strategy games, 2009.

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.

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.

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.

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.

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.

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.

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

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.

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
```