

## Lecture Notes on Game Playing

*Alexander Rush***Contents**

<b>1</b>	<b>Board 0</b>	<b>2</b>
<b>2</b>	<b>Board 0.5</b>	<b>2</b>
<b>3</b>	<b>Board 1</b>	<b>3</b>
<b>4</b>	<b>Board 2</b>	<b>3</b>
<b>5</b>	<b>Board 3</b>	<b>3</b>
<b>6</b>	<b>Board 4</b>	<b>4</b>
<b>7</b>	<b>Board 5</b>	<b>4</b>
<b>8</b>	<b>Board 5</b>	<b>5</b>
<b>9</b>	<b>Board 6</b>	<b>5</b>
9.1	Game Model . . . . .	5
<b>10</b>	<b>Board 7</b>	<b>5</b>
<b>11</b>	<b>Board 8</b>	<b>6</b>
<b>12</b>	<b>Board 9</b>	<b>6</b>
<b>13</b>	<b>Board 9</b>	<b>6</b>
<b>14</b>	<b>Board 10</b>	<b>7</b>
<b>15</b>	<b>Board 11</b>	<b>7</b>
<b>16</b>	<b>Board 12</b>	<b>8</b>
<b>17</b>	<b>Board 13</b>	<b>8</b>
<b>18</b>	<b>Board 14</b>	<b>8</b>
18.1	Alpha-Beta Pruning . . . . .	8

19	Board 15	9
20	Board 15	9
21	Board 16	9
22	Board 17	10
23	Board 18	10
24	Board 19	10
25	Board 20	11
26	Board 21	11
	26.1 Heuristics . . . . .	11
27	Board 22	13
	27.1 Expecti-Max . . . . .	13
28	Board 23	13
29	Board 23	13
30	Board 24	13

## 1 Board 0

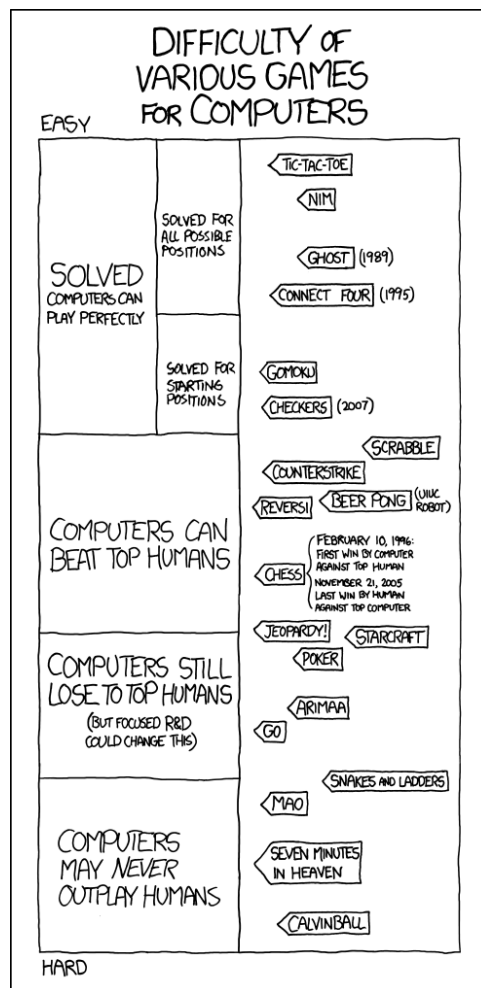
Show starcraft

[https://www.youtube.com/watch?v=LjSXj4cb\\_Yo](https://www.youtube.com/watch?v=LjSXj4cb_Yo)

## 2 Board 0.5

Heuristic examples:

### 3 Board 1



### 4 Board 2

Show XKCD In this lecture we will use examples from Tic-Tac-Toe, later in the term we will talk about Jeopardy and StarCraft. Backgammon is also a particularly interesting example that we'll discuss briefly. If you're interested MIT has run a great IAP tournaments in the past focusing on Poker and StarCraft. There used to be a substantial reward for solving Go. However, Seven Minutes in Heaven is beyond the scope of this course.

### 5 Board 3

History of Chess in AI

- 1948 - Alan Turing TuroChamp

- 1949-1950 - Claude Shannon ([http://www.computerhistory.org/chess/full\\_record.php?iid=stl-430b9bbe92716](http://www.computerhistory.org/chess/full_record.php?iid=stl-430b9bbe92716)) 1949 by Claude Shannon
- 
- 1959 - John McCarthy and students, Allen Newell, Cliff Shaw, Herbert Simon (central AI figure)
- 1973 - Switch to Type-A (brute-force) systems Chess 4.0
- 1997 - IBM Deep Blue (8-12 depth, 200million moves)
- 2002 2006 - Commercially available (Deep Fritz) draws with Kramnick
- Junior - Search for interesting moves.
- Switch to Go.
- Current: Centaur chess.
  - 6.176 - <http://mitpokerbots.com/>
  - 6.370 - <https://www.battlecode.org/>

## 6 Board 4

Vocabulary:

- instead of minimizing cost, obtain **utility**
- Utility is only given at **terminal** states
- Assume games are **zero-sum**
- **partial information perfect information**, i.e. both players see state of the game. Partial: cards, stratego, starcraft, Perfect
- **deterministic** versus **stochastic**, i.e. both players see state of the game. cards, backgammon, etc.
- **two-player** versus **multiplayer** chess, checkers versus, settlers, poker  
Min/ Max Two players

Chess and go.

## 7 Board 5

Name (AIMA)	Type	Description
State space	$\mathcal{S}$	All states of the game, e.g. unique board positions
Action space	$\mathcal{A}$	All actions of the game i.e. moves
Actions	$\text{ACT} : \mathcal{S} \mapsto 2^{\mathcal{A}}$	Actions at state, i.e. available moves.
Transition model	$\text{RES} : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{S}$	Update the state after a move.
Initial state	$s_0 \in \mathcal{S}$	The starting state of the game.
Player	$\text{PLAYER} : \mathcal{S} \rightarrow \{\text{Min}, \text{Max}\}$	The current turn.
Term test	$\text{TERM} : \mathcal{S} \mapsto \{0, 1\}$	Is the state is terminal, i.e. is the game over.
Utility	$\text{UTILITY} : \mathcal{S} \rightarrow \mathbb{R}$	The utility of (terminal) state.

We will be treating game playing as a search problem using a similar abstract specification. However there are several additional elements that we need to include.

Chess:  
Settlers of Catan:  
Starcraft

## 8 Board 5

How do we treat a current state in the game? What will Utility be.

Assumption: **Rationality**: Each agent acts simply to gain utility.

The key challenge of adversarial search is that the agent does not control the decisions of its opponent. Therefore to behave rationally i.e. to minimize or maximize utility, the agent must model its opponents decisions.

Generally we do this by assuming that the opponent will behave rationally as well, so if we are minimizing utility, we assume the opponent will act to maximize their utility. This leads to the Minimax formula for computing the utility of a nonterminal game state.

## 9 Board 6

### 9.1 Game Model

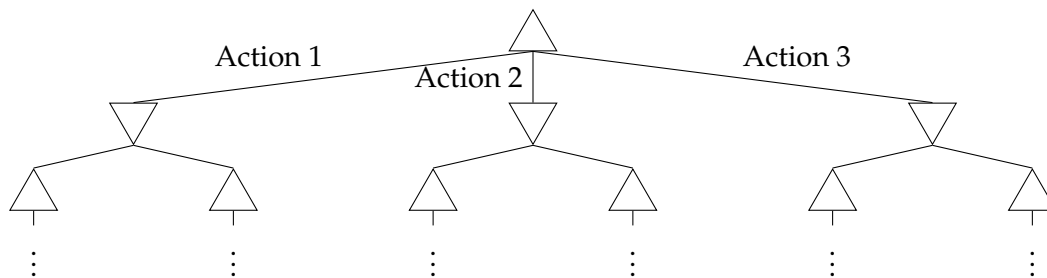
Under a fully rational game

$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERM}(s) \\ \max_{a \in \text{ACT}(s)} \text{MINIMAX}(\text{RES}(s, a)) & \text{if PLAYER}(s) = \text{Max} \\ \min_{a \in \text{ACT}(s)} \text{MINIMAX}(\text{RES}(s, a)) & \text{if PLAYER}(s) = \text{Min} \end{cases}$$

## 10 Board 7

### Game Tree

We can expand this function into a data-structure known as a **game tree**. Each node in this tree represents a move by Max or Min. The triangles pointing up indicate that the node is maximizing over its children, and triangles pointing down indicate that the node is minimizing over its children.



## 11 Board 8

### Tic-Tac-Toe

Let's now consider one of the simplest zero-sum, perfect-information games, Tic-Tac-Toe. Consider the model for the following game state:

Max: O

Min: X

O		
O	X	X
X	X	O

## 12 Board 9

**Question 1** What is the model state at this stage of tic-tac-toe?

Name	Type	Description
Actions	⊗ $ACT(s) = \{ OTop, OTopRight \}$	⊗ There are currently two actions available to play.
Player	⊗ $PLAYER(s) = \text{Min}$	⊗ We will assume that O is minimizing utility and it is her turn to play.
Term test	⊗ $TERM(s) = 0$	⊗ This is not a terminal state. We have yet to win or draw
Utility	⊗ $UTILITY$	⊗ The utility function at a terminal state will be 1 for X, -1 for O, 0 for a draw.

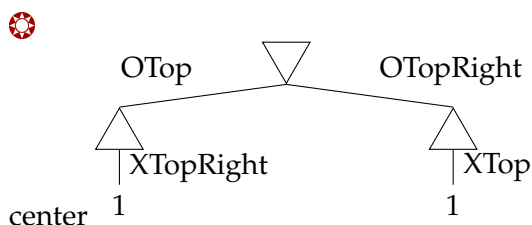
Next we can ask what is the utility that the Min player should assign to this state. We can do this by expanding out the recursive MINIMAX definition. This gives:

### 13 Board 9

$$MINIMAX(s) = \min \left\{ \begin{array}{l} \max \{ UTILITY(RES(RES(s, OTop)), XTopRight) \}, \\ \max \{ UTILITY(RES(RES(s, OTopRight)), XTop) \} \end{array} \right\}$$

Since either way O plays, it leads to an X win, this utility of this state is 1.

**Question 2** Can we also write the same calculation as a game tree?

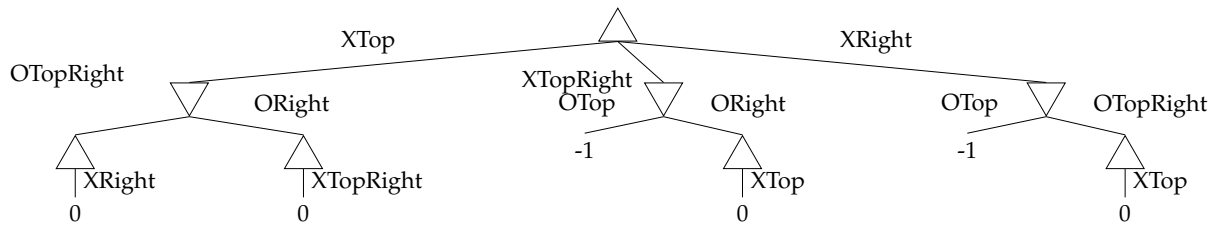


### 14 Board 10

Now let's consider a slightly more interesting position from X's perspective

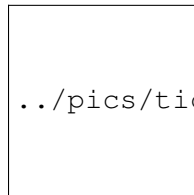
O		
X	O	
X	O	X

**Question 3** Let's write the same calculation as a game tree.



The max player (X) can at best play for a draw here. Clearly the right play is XTop which gives the utility of 0.

Note though that for these problems we are only considering a branching factor of at most 3, and a depth of 2. In practice the tree grows very large, very quickly. AIMA has a nice graphic showing a partial expansion of a game tree for tic-tac-toe from scratch.



## 15 Board 11

- Graph Search for games
  - Depth-first search
  - Tree Search

In particular, we utilize uninformed depth-first tree search where are terminal states are goal states.

**Question 4** *Why do we not need to use graph search for games? Why not use uniform-cost search?*

- DFS with tree search has great memory efficiency.

In this situation, DFS has clear advantages over BFS in terms of memory-efficiency. Because of the nature of game playing, we do not have to worry about maintaining the costly expand list.

## 16 Board 12

Note also that instead of explicitly using a stack, we can implement DFS using recursion. Each recursive call to MINIMAX simulates a pushing and popping from the stack. This leads to a very simple implementation of Minimax Search.



---

```

1: procedure MINIMAX( $s$ )
2:   if TERM( $s$ ) then return UTILITY( $s$ )
3:   else if PLAYER( $s$ ) = Max then
4:      $v \leftarrow -\infty$ 
5:     for  $a \in \text{ACT}(s)$  do
6:        $v \leftarrow \max\{v, \text{MINIMAX}(\text{RES}(s, a))\}$ 
7:   else if PLAYER( $s$ ) = Min then
8:      $v \leftarrow \infty$ 
9:     for  $a \in \text{ACT}(s)$  do
10:       $v \leftarrow \min\{v, \text{MINIMAX}(\text{RES}(s, a))\}$ 
11:  return  $v$ 

```

---

## 17 Board 13

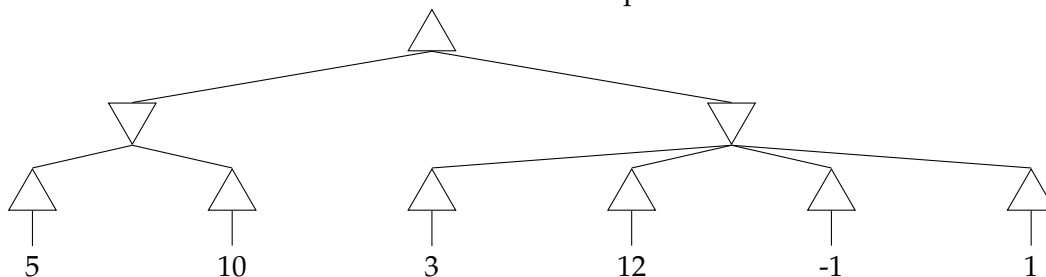
**Question 5** What is the space and time complexity of this algorithm?

Again let  $m$  be the depth of the deepest terminal state and  $b$  be the max branching factor. This algorithm need to enumerate each possible state to this point giving a run-time of  $O(b^m)$ . However at any one time, it only needs memory for all actions at each state along the path to the deepest goal. This gives a memory complexity of  $O(bm)$ .

## 18 Board 14

### 18.1 Alpha-Beta Pruning

We can further speed up game-playing search by exploiting some properties about the game tree. One nice property of the max (min) operator, is that if we know after seeing  $i$  elements of the set that known of the future elements will be greater (less) then it is not necessary to compute these values at all. We can therefore **short circuit** the computation and return.



## 19 Board 15

**Question 6** If instead of  $-1$  the utility was  $-1000$ , would anything change? What if instead 5 was equal to 2?

What happened in this case was that the already computed value 5 already dominated the value of 3 in the min. Since 3 is already lower than 5 there is no need to keep on calculating the rest of the values.

## 20 Board 15

**Alpha-beta pruning** speeds up minimax search while still given the exact result..

$$\max\{\alpha, \dots, \min\{\dots, v, \dots, v'\}\}$$

Computed the value of  $\alpha$  and  $v$ , do we need to compute the rest of the min including  $v'$ . Consider the two cases:

1.  $v > \alpha$ ; inner min may end with a value higher than  $\alpha$
2.  $v \leq \alpha$ ; inner min cannot change outer max, value of  $v'$  is uninformative.

Examples the **alpha** value was  $\alpha = 5$  and  $v = 3$ .

**Generalize:**

$$\max\{\dots, \alpha, \dots, \min\{\max\{\dots \min\{\dots, v, \dots, v'\}\}\}, \dots\}$$

If there is a future value  $v'$  with  $v' \leq v$  it may be selected by the inner min. However we know  $v' < v \leq \alpha$ .

so it will not be selected by the outer max. The bigger  $\alpha$  we have, the more likely we can apply this pruning, so we maintain the largest  $\alpha$  seen at ancestor max node.

## 21 Board 16

**Reverse trick**

$$\ominus \min\{\dots, \beta, \dots, \max\{\min\{\dots \min\{\dots, v, \dots, v'\}\}\}, \dots\}$$

## 22 Board 17

The modified Minimax search algorithm is shown below. The upside of this method is that it yields a significant speed-up in practice with very minimal bookkeeping and no effect on the result. A rare combination.

Doesn't change complexity in the worst case, practically gives a 2x speed up.  
ordering moves matters now!

---

**Require:**  $\alpha$  is highest value of ancestor max-node,  $\beta$  is lowest value of ancestor min-node

```

1: procedure ALPHABETAMINIMAX( $s, \alpha, \beta$ )
2:   if TERM( $s$ ) = 1 then return UTILITY( $s$ )
3:   else if PLAYER( $s$ ) = Max then
4:      $v \leftarrow -\infty$ 
5:      $v \leftarrow \text{⊛}$ 
6:     for  $a \in \text{ACT}(s)$  do
7:        $v \leftarrow \text{⊛} \max\{v, \text{ALPHABETAMINIMAX}(\text{RES}(s, a), \alpha, \beta)\}$ 
8:       if  $v \geq \beta$  then return  $v$ 
9:        $\alpha \leftarrow \max\{\alpha, v\}$ 
10:  else if PLAYER( $s$ ) = Min then
11:     $v \leftarrow \infty$ 
12:    for  $a \in \text{ACT}(s)$  do
13:       $v \leftarrow \text{⊛} \min\{v, \text{ALPHABETAMINIMAX}(\text{RES}(s, a), \alpha, \beta)\}$ 
14:      if  $\text{⊛} v \leq \alpha$  then return  $v$ 
15:       $\text{⊛} \beta \leftarrow \min\{\beta, v\}$ 
16:  return  $v$ 

```

---

## 23 Board 18

### Chess

Average branching factor: 35

Average depth: 40

(apparently there was a 277 move game once)

$O(35^{40})$  is cartoonishly large (actually worse!)

57905761067641178540192733082440108773880638182163238525390625

## 24 Board 19

Depth-limited scoring function

Heuristic  $h$

$$\text{H-MINIMAX}(s, d) = \begin{cases} \text{UTILITY}(s) & \text{if TERM}(s) \\ h(s) & \text{if } d > \text{limit} \\ \max_{a \in \text{ACT}(s)} \text{H-MINIMAX}(\text{RES}(s, a), d + 1) & \text{if PLAYER}(s) = \text{Max} \\ \min_{a \in \text{ACT}(s)} \text{H-MINIMAX}(\text{RES}(s, a), d + 1) & \text{if PLAYER}(s) = \text{Min} \end{cases}$$

No notion of admissable or consistent here.

$h : \mathcal{S} \mapsto \mathbb{R}$

## 25 Board 20

Heuristics Perhaps the simplest heuristic is the value system taught to beginners learning to play chess. This system simply assigns value to the **material remaining** on the board.






Symbol					
Piece	pawn	knight	bishop	rook	queen
Value	1	3	3	5	9

## 26 Board 21

### 26.1 Heuristics

Generally game playing heuristics estimate how good a position is for each player. Note that unlike search heuristics we will not make any attempt to maintain admissibility or consistency here, although similar conditions do exist.

However, note that because this is just an approximation and ignores so much of the other complications of chess, there have been many, many other heuristics for this problem. (Wikipedia is amazing.) I am not a chess player, but it is pretty remarkable to me that not even the relative values of bishop and knight are consistent.

					Source	Date	Comment
3.1	3.3	5.0	7.9	2.2	Sarratt?	1813	(rounded) pawns vary from 0.7 to 1.3 <sup>[1]</sup>
3.05	3.50	5.48	9.94		Philidor	1817	also given by Staunton in 1847 <sup>[2]</sup>
3	3	5	10		Peter Pratt	early 19th century	(Hooper & Whyld 1992:439)
3.5	3.5	5.7	10.3		Bilguer	1843	(rounded) (Hooper & Whyld 1992:439) <sup>[3]</sup>
3	3	5	9-10	4	Lasker	1934	<sup>[4]</sup> (Lasker 1934:73)
3½	3½	5½	10		Euwe	1944	(Euwe & Kramer 1994:11)
3½	3½	5	8½	4	Lasker	1947	(rounded) Kingside rooks and bishops are valued more, queenside ones less <sup>[5]</sup>
3	3+	5	9		Horowitz	1951	The bishop is "3 plus small fraction" (Horowitz 1951:11), (Horowitz & Rothenberg 1963:36)
3½	3½+	5	10	4	Evans	1958	Bishop is 3¾ if in the bishop pair <sup>[6]</sup> (Evans 1958:77,80)
3½	3½	5	9½		Styeklov (early Soviet chess program)	1961	(Soltis 2004:6) (Levy & Newborn 1991:45)
3	3¼	5	9		Fischer	1972	(Fischer, Mosenfelder & Margulies 1972:14)
3	3	4½	8½		European Committee on Computer Chess, Euwe	1970s	(Brace 1977:236)
3	3	4½	9		Garry Kasparov	1986	(Kasparov 1986:9)
3	3	5	9-10		Soviet chess encyclopedia	1990	A queen equals three minor pieces or two rooks (Hooper & Whyld 1992:439)
3¼	3¼	5	9¾		Kaufman	1999	Add ½ point for the bishop pair <sup>[7]</sup> (Kaufman 1999)
3.20	3.33	5.10	8.80		Berliner	1999	plus adjustments for openness of position, rank & file (Berliner 1999:14–18)
3½	3½	5	9		Kurzdorfer	2003	(Kurzdorfer 2003:94)
3	3	4½	9		another popular system		(Soltis 2004:6)
4	3½	7	13½	4	used by a computer		Two bishops are worth more (Hooper & Whyld 1992:439)
2.4	4.0	6.4	10.4	3.0	Yevgeny Gik		based on average mobility; Soltis (2004:10–12) pointed out problems with this type of analysis

Another method that is used is brute force calculation of certain positions. In chess this often means utilizing an end-game solver to play out the results of a large set of standard positions (such as a kings and pawns ending) . Given that each extra depth is a multiplicative cost, having an endgame database can turn a failed search into a successful one.

- LandmarkLowerBound(S,G) — StarCraft’s tech tree imposes many prerequisites on actions. These actions are known in the search literature as landmarks. Given this sequence of non-concurrent landmark actions, we sum the individual durations of actions not yet created to form an admissible lower bound for our search.

- ResourceGoalBound(S,G) — Summing the total consumed resource cost of units in a goal gives us a lower bound on the resources required to construct the goal optimally. Performing a quick search to determine the makespan of producing only these resources is an admissible heuristic.

Monte-carlo style- simulate out the remaining game

Based on playouts.

## 27 Board 22

### 27.1 Expecti-Max

So far we have consider only deterministic games with perfect information. We will end by discussing games like Backgammon, which have perfect information but incorporates randomness. Each turn in backgammon consists of two steps:

- Roll two dies
- Select checkers to move based on dice values.

## 28 Board 23

Our focus will be on the question of how to model our opponent Min's action. Since we don't know what value they will roll, we could be pessimistic and assume that they get the best possible roll (min over rolls) or optimistic assume they get the worst possible roll (max over rolls).

Instead, we will directly model the randomness of the roll. We treat the dice roll as a third player Exp and the value of the dice as an action  $a$ . Given the state of the world  $s$ , we say the probability of Exp taking action  $a$  is given as  $P(a|s)$ .

In the case of Backgammon, there are 36 possible actions corresponding to each pair of dice rolls, each with probability  $\frac{1}{36}$ . The turn order then alternates: Exp, Min, Exp, Max, Exp, Min, ... Once we explicitly incorporate the last roll into the state  $s$ , the moves of Max and Min become deterministic.

## 29 Board 23

$$\mathbb{E}_{x|y}[f(x)] = \sum_x p(x|y)f(x)$$

In the case of the utility of a state, this gives us a method for computing the expected utility of an Exp state.

$$\mathbb{E}_{a|s}[\text{UTILITY}(\text{RES}(s, a))] = \sum_{a \in \text{ACT}(s)} p(a|s) \text{UTILITY}(\text{RES}(s, a))$$

This formula can then be directly plugged into minimax, to yield an algorithm known as expectimax

## 30 Board 24

$$\text{E-MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERM}(s) \\ \max_{a \in \text{ACT}(s)} \text{E-MINIMAX}(\text{RES}(s, a)) & \text{if PLAYER}(s) = \text{Max} \\ \min_{a \in \text{ACT}(s)} \text{E-MINIMAX}(\text{RES}(s, a)) & \text{if PLAYER}(s) = \text{Min} \\ \sum_{a \in \text{ACT}(s)} p(a|s) \text{E-MINIMAX}(\text{RES}(s, a)) & \text{if PLAYER}(s) = \text{Exp} \end{cases}$$