Min-Max Algorithm

```
max-player(s)
    for a in $A_s$
        s’ <— Result (s, a)
        v(a) <— min-eval(s’,2)
    return value = max v(a), action = argmax v(a)
```

```
min-eval(s,d):
    if d > depth; return eval(s)
    if s is terminal; return eval(s)
    v <— +infinity
    for a in $A_s$
        s’ <— Result (s, a)
        v <—min { v, max-eval(s’,d+1) }
    return v
```

```
max-eval(s,d):
    if d > depth; return eval(s)
    if s is terminal; return eval(s)
    v <— -infinity
    for a in $A_s$
        s’ <— Result (s, a)
        v <—max { v, min-eval(s’,d+1)
    return v
```

Alpha Beta Pruning

```
max-player(s)
    alpha = -infinity, beta = +infinity
    for a in $A_s$
        s’ = Result (s, a)
        v(a) <— min-eval(s’,2,alpha,beta)
        alpha <— max {alpha, v(a) }
    return value = max v(a), action = argmax v(a)
```

```
min-eval(s,d,alpha,beta):
    $beta_{local}$ <— beta
    if d > depth; return eval(s)
    if s is terminal; return eval(s)
    v <— +infinity
    for a in $A_s$
        s’ = Result (s, a)
        v <—min { v, max-eval(s’,d+1,alpha,$beta_{local}$) }
        if v <= alpha; then return v
        $beta_{local}$ <— min {$beta_{local}$ , v }
    return v
```

```
max-eval(s,d,alpha,beta):
    $alpha_{local}$ <— alpha
    if d > depth; return eval(s)
    if s is terminal; return eval(s)
    v <— -infinity
    for a in $A_s$
        s’ = Result (s, a)
        v <—max { v, min-eval(s’,d+1,$alpha_{local}$,beta) }
        if v >= beta; then return v
        $alpha_{local}$ <— max {$alpha_{local}$ , v }
    return v
```