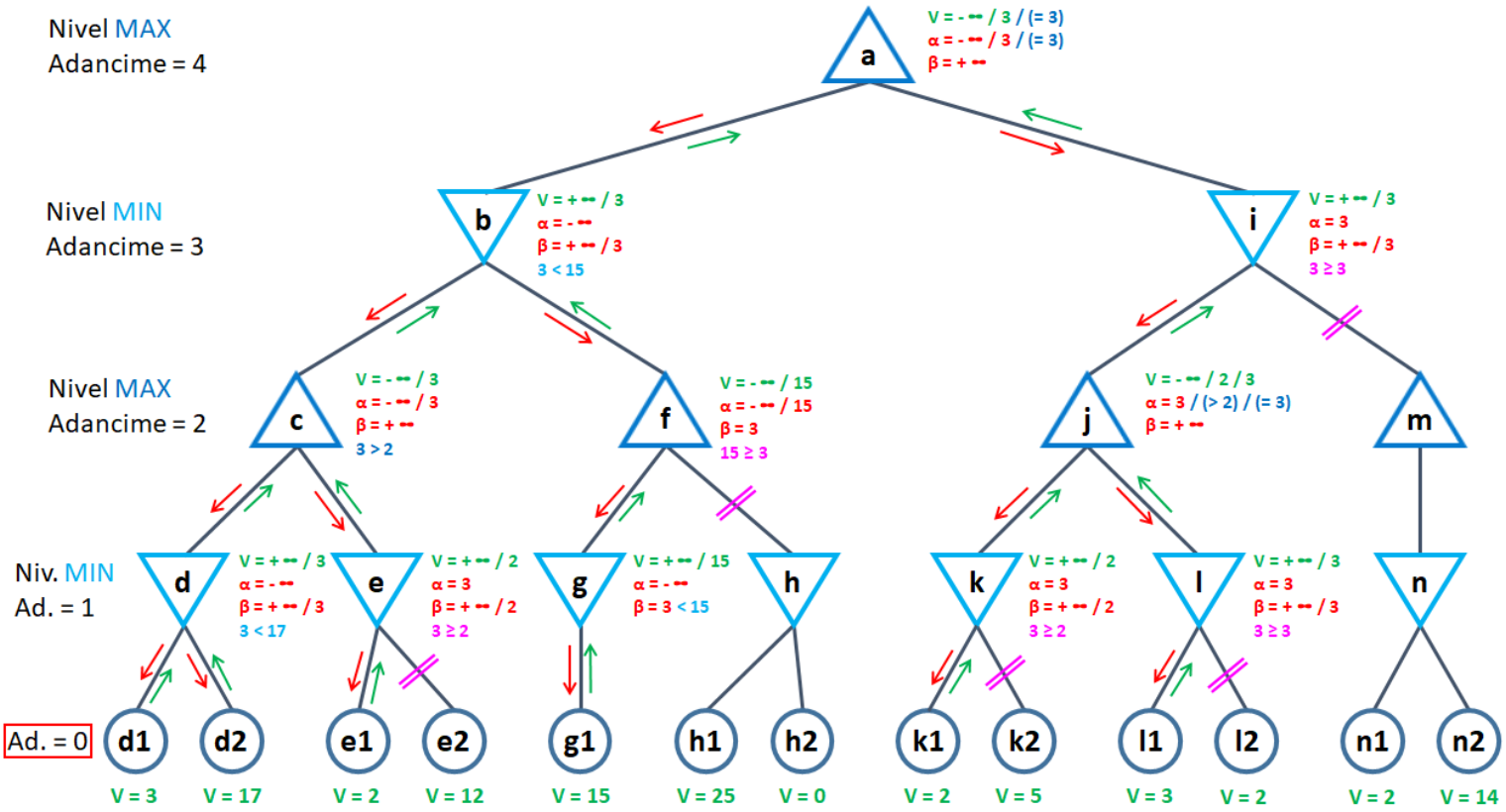
→ *Exemplu detaliat pentru algoritmul Alpha-Beta:*



Apel general: alpha_beta(alpha, beta, Stare(nod, tip_nod, adancime)).

Se initializeaza intervalul cu **alpha = - inf** si **beta = + inf**, iar **adancime = 4**.
[Apel 1] alpha_beta(-inf, +inf, Stare(“**a**”, JMAX, 4))
Se porneste din nodul „a” de tip **MAX**, caruia i se initializeaza scorul **V(„a”) = - inf**.
Se calculeaza: mutari(“a”) = [“b”, “i”] # apeluri 2 si 12

[Apel 2] alpha_beta(-inf, inf, Stare(“**b**”, JMIN, 3))
Se continua cu primul fiu al lui „a”, adica nodul „b” de tip **MIN**:
- care primeste intervalul de la parinte: **alpha = -inf, beta = inf**
- si i se initializeaza scorul **V(„b”) = + inf**
Se calculeaza: mutari(“b”) = [“c”, “f”] # apeluri 3 si 9

[Apel 3] alpha_beta(-inf, inf, Stare(“**c**”, JMAX, 2))
Se continua cu primul fiu al lui “b”, adica nodul “c” de tip MAX:
- care primeste intervalul de la parinte: **alpha = -inf, beta = inf**
- si i se initializeaza scorul **V(„c”) = - inf**
Se calculeaza: mutari(“c”) = [“d”, “e”] # apeluri 4 si 7

[Apel 4] alpha_beta(-inf, inf, Stare(“**d**”, JMIN, 1))
Se continua cu primul fiu al lui “c”, adica nodul “d” de tip MIN:
- care primeste intervalul de la parinte: **alpha = -inf, beta = inf**
- si i se initializeaza scorul **V(„d”) = + inf**
Se calculeaza: mutari(“d”) = [“d1”, “d2”] # apeluri 5 si 6

[Apel 5] alpha_beta(-inf, inf, Stare(“**d1**”, JMAX, 0))
Se continua cu primul fiu al lui “d”, adica nodul ”d1”.
Pentru ca am ajuns la **adancime = 0**
=> se calculeaza **V(“d1”) = 3** (cu functia “estimeaza_scor”)
Functia alpha_beta returneaza starea “d1” cu scorul 3.
 # iesire din Apel 5 => continuare apel 4

[apel 4 continuare1] Revenim in starea nodului “d” de tip **MIN**.
Se verifica daca V(fiu = “d1”) poate ***minimiza*** valorile **V** si **beta** pt tata = ”d”.
1) Daca V(“d1”) < V(“d”): # 3 < (+inf)
   - V(“d”) = V(“d1”) # **V(“d”) = 3**
   - **fiu_ales(“d”) = “d1”**
2) Daca V(“d1”) < beta(“d”) # 3 < (+inf)
   - beta(“d”) = V(“d1”) # **beta(“d”) = 3**
   - Verificam daca alpha(“d”) ≥ beta(“d”): # (-inf) ≥ 3 => False

[Apel 6] alpha_beta(-inf, 3, Stare(“**d2**”, JMAX, 0))
Se continua cu al doilea fiu al lui “d”, adica nodul “d2”.
Pentru ca am ajuns la **adancime = 0**
=> se calculeaza **V(“d2”) = 17** (cu functia “estimeaza_scor”)
Functia alpha_beta returneaza starea “d2” cu scorul 17.
 # iesire din Apel 6 => continuare apel 4

[apel 4 continuare2] Revenim in starea nodului “d” de tip **MIN**.
Se verifica daca V(fiu = “d2”) poate ***minimiza*** valorile **V** si **beta** pt tata = ”d”.
1) Daca V(“d2”) < V(“d”): # 17 < 3 => False
2) Daca V(“d2”) < beta(“d”): # 17 < 3 => False
Functia alpha_beta returneaza starea “d” cu scorul 3 si fiu_ales = ”d1”.
 # iesire din Apel 4 => continuare apel 3

[apel 3 continuare1] Revenim in starea nodului “c” de tip **MAX**.
Se verifica daca V(fiu = “d”) poate ***maximiza*** valorile **V** si **alpha** pt tata = ”c”.
1) Daca V(“d”) > V(“c”): # 3 > (-inf)
   - V(“c”) = V(“d”) # **V(“c”) = 3**
   - **fiu_ales(“c”) = “d”**
2) Daca V(“d”) > alpha(“c”) # 3 > (-inf)
   - alpha(“c”) = V(“d”) # **alpha(“c”) = 3**
   - Verificam daca alpha(“c”) ≥ beta(“c”): # 3 ≥ (+inf) => False

[Apel 7] alpha_beta(3, inf, Stare(“**e**”, JMIN, 1))
Se continua cu al doilea fiu al lui “c”, adica nodul “e” de tip MIN:
- care primeste intervalul de la parinte: **alpha = 3, beta = +inf**
- si i se initializeaza scorul **V(„e”) = + inf**
Se calculeaza: mutari(“e”) = [“e1”, “e2”] # apel 8 si retezare “e2”

[Apel 8] alpha_beta(3, inf, Stare(“**e1**”, JMAX, 0))
Se continua cu primul fiu al lui “e”, adica nodul ”e1”.
Pentru ca am ajuns la **adancime = 0**
=> se calculeaza **V(“e1”) = 2** (cu functia “estimeaza_scor”)
Functia alpha_beta returneaza starea “e1” cu scorul 2.
 # iesire din Apel 8 => continuare apel 7

[apel 7 continuare] Revenim in starea nodului “e” de tip **MIN**.
Se verifica daca V(fiu = “e1”) poate ***minimiza*** valorile **V** si **beta** pt tata = ”e”.
1) Daca V(“e1”) < V(“e”): # 2 < (+inf)
   - V(“e”) = V(“e1”) # **V(“e”) = 2**
   - **fiu_ales(“e”) = “e1”**
2) Daca V(“e1”) < beta(“e”) # 2 < (+inf)
   - beta(“e”) = V(“e1”) # **beta(“e”) = 2**
   - Verificam daca alpha(“e”) ≥ beta(“e”): # 3 ≥ 2 => **True**
     => ***Retezam*** restul fiilor nodului “e” (nodul “e2”).
Functia alpha_beta returneaza starea “e” cu scorul 2 si fiu_ales = ”e1”.
 # iesire din Apel 7 => continuare apel 3

[apel 3 continuare2] Revenim in starea nodului “c” de tip **MAX**.
Se verifica daca V(fiu = “e”) poate ***maximiza*** valorile **V** si **alpha** pt tata = ”c”.
1) Daca V(“e”) > V(“c”): # 2 > 3 => False
2) Daca V(“e”) > alpha(“c”) # 2 > 3 => False
Functia alpha_beta returneaza starea “c” cu scorul 3 si fiu_ales = “d”.
 # iesire din Apel 3 => continuare apel 2

[apel 2 continuare1] Revenim in starea nodului “b” de tip **MIN**.
Se verifica daca V(fiu = “c”) poate ***minimiza*** valorile **V** si **beta** pt tata = ”b”.
1) Daca V(“c”) < V(“b”): # 3 < (+inf)
- V(“b”) = V(“c”) # **V(“b”) = 3**
- **fiu_ales(“b”) = “c”**

2) Daca V(“c”) < beta(“b”) # 3 < (+inf)
- beta(“b”) = V(“c”) # **beta(“b”) = 3**
- Verificam daca alpha(“b”) ≥ beta(“b”): # (-inf) ≥ 3 => False

[Apel 9] alpha_beta(-inf, 3, Stare(“**f**”, JMAX, 2))
Se continua cu al doilea fiu al lui “b”, adica nodul “f”.
- care primeste intervalul de la parinte: **alpha = -inf, beta = 3**
- si i se initializeaza scorul **V(„f”) = - inf**
Se calculeaza: mutari(“f”) = [“g”, “h”] # apel 10 si retezare “h”

[Apel 10] alpha_beta(-inf, 3, Stare(“**g**”, JMIN, 1))
Se continua cu primul fiu al lui “f”, adica nodul “g” de tip MIN:
- care primeste intervalul de la parinte: **alpha = -inf, beta = 3**
- si i se initializeaza scorul **V(„g”) = + inf**
Se calculeaza: mutari(“g”) = [“g1”] # apel 11

[Apel 11] alpha_beta(-inf, 3, Stare(“**g1**”, JMAX, 0))
Se continua cu primul fiu al lui “g”, adica nodul ”g1”.
Pentru ca am ajuns la **adancime = 0**
=> se calculeaza **V(“g1”) = 15** (cu functia “estimeaza_scor”)
Functia alpha_beta returneaza starea “g1” cu scorul 15.
 # iesire din Apel 11 => continuare apel 10

[apel 10 continuare] Revenim in starea nodului “g” de tip **MIN**.
Se verifica daca V(fiu = “g1”) poate ***minimiza*** valorile **V** si **beta** pt tata = ”g”.
1) Daca V(“g1”) < V(“g”): # 15 < (+inf)
- V(“g”) = V(“g1”) # **V(“g”) = 15**
- **fiu_ales(“g”) = “g1”**

2) Daca V(“g1”) < beta(“g”) # 15 < 3 => False
Functia alpha_beta returneaza starea “g” cu scorul 15 si fiu_ales = “g1”.
 # iesire din Apel 10 => continuare apel 9

[apel 9 continuare] Revenim in starea nodului “f” de tip **MAX**.
Se verifica daca V(fiu = “g”) poate ***maximiza*** valorile **V** si **alpha** pt tata = ”f”.
1) Daca V(“g”) > V(“f”): # 15 > (-inf)
- V(“f”) = V(“g”) # **V(“f”) = 15**
- **fiu_ales(“f”) = “g”**

2) Daca V(“g”) > alpha(“f”) # 15 > (-inf)
- alpha(“f”) = V(“g”) # **alpha(“f”) = 15**
- Verificam daca alpha(“f”) ≥ beta(“f”): # 15 ≥ 3 => **True**
  => ***Retezam*** restul fiilor nodului “f” (subarborele de radacina “h”).

Functia alpha_beta returneaza starea “f” cu scorul 15 si fiu_ales = “g”.
 # iesire din Apel 9 => continuare apel 2

[apel 2 continuare2] Revenim in starea nodului “b” de tip **MIN**.
Se verifica daca V(fiu = “f”) poate ***minimiza*** valorile **V** si **beta** pt tata = ”b”.
1) Daca V(“f”) < V(“b”): # 15 < 3 => False
2) Daca V(“f”) < beta(“b”) # 15 < 3 => False
Functia alpha_beta returneaza starea “b” cu scorul 3 si fiu_ales = “c”.
 # iesire din Apel 2 => continuare apel 1

[apel 1 continuare1] Revenim in starea nodului “a” de tip **MAX**.
Se verifica daca V(fiu = “b”) poate ***maximiza*** valorile **V** si **alpha** pt tata = ”a”.
1) Daca V(“b”) > V(“a”): # 3 > (-inf)
- V(“a”) = V(“b”) # **V(“a”) = 3**
- **fiu_ales(“a”) = “b”**

2) Daca V(“b”) > alpha(“a”) # 3 > (-inf)
- alpha(“a”) = V(“b”) # **alpha(“a”) = 3**
- Verificam daca alpha(“a”) ≥ beta(“a”): # 3 ≥ (+inf) => False

[Apel 12] alpha_beta(3, inf, Stare(“**i**”, JMIN, 3))
Se continua cu al doilea fiu al lui “a”, adica nodul “i” de tip MIN:
- care primeste intervalul de la parinte: **alpha = 3, beta = +inf**
- si i se initializeaza scorul **V(„i”) = + inf**
Se calculeaza: mutari(“i”) = [“j”, “m”] # apel 13 si retezare “m”

[Apel 13] alpha_beta(3, inf, Stare(“**j**”, JMAX, 2))
Se continua cu primul fiu al lui “i”, adica nodul “j” de tip MAX:
- care primeste intervalul de la parinte: **alpha = 3, beta = inf**
- si i se initializeaza scorul **V(„j”) = - inf**
Se calculeaza: mutari(“j”) = [“k”, “l”] # apeluri 14 si 16

[Apel 14] alpha_beta(3, inf, Stare(“**k**”, JMIN, 1))
Se continua cu primul fiu al lui “j”, adica nodul “k” de tip MIN:
- care primeste intervalul de la parinte: **alpha = 3, beta = inf**
- si i se initializeaza scorul **V(„k”) = + inf**
Se calculeaza: mutari(“k”) = [“k1”, “k2”] # apel 15 si retezare “k2”

[Apel 15] alpha_beta(3, inf, Stare(“**k1**”, JMAX, 0))
Se continua cu primul fiu al lui “k”, adica nodul ”k1”.
Pentru ca am ajuns la **adancime = 0**
=> se calculeaza **V(“k1”) = 2** (cu functia “estimeaza_scor”)
Functia alpha_beta returneaza starea “k1” cu scorul 2.
 # iesire din Apel 15 => continuare apel 14

[apel 14 continuare] Revenim in starea nodului “k” de tip **MIN**.
Se verifica daca V(fiu = “k1”) poate ***minimiza*** valorile **V** si **beta** pt tata = ”k”.
1) Daca V(“k1”) < V(“k”): # 2 < (+inf)
- V(“k”) = V(“k1”) # **V(“k”) = 2**
- **fiu_ales(“k”) = “k1”**

2) Daca V(“k1”) < beta(“k”) # 2 < (+inf)
- beta(“k”) = V(“k1”) # **beta(“k”) = 2**
- Verificam daca alpha(“k”) ≥ beta(“k”): # 3 ≥ 2 => **True**
  => ***Retezam*** restul fiilor nodului “k” (nodul “k2”).

Functia alpha_beta returneaza starea “k” cu scorul 2 si fiu_ales = “k1”.
 # iesire din Apel 14 => continuare apel 13

[apel 13 continuare1] Revenim in starea nodului “j” de tip **MAX**.
Se verifica daca V(fiu = “k”) poate ***maximiza*** valorile **V** si **alpha** pt tata = ”j”.
1) Daca V(“k”) > V(“j”): # 2 > (-inf)
- V(“j”) = V(“k”) # **V(“j”) = 2**
- **fiu_ales(“j”) = “k”**

2) Daca V(“k”) > alpha(“j”) # 2 > 3 => False

[Apel 16] alpha_beta(3, inf, Stare("**l**", JMIN, 1))
Se continua cu al doilea fiu al lui "j", adica nodul "l" de tip MIN:
- care primeste intervalul de la parinte: **alpha = 3, beta = +inf**
- si i se initializeaza scorul **V(„l") = + inf**
Se calculeaza: mutari("l") = ["l1", "l2"] # apel 17 si retezare "l2"

[Apel 17] alpha_beta(3, inf, Stare("**l1**", JMAX, 0))
Se continua cu primul fiu al lui "l", adica nodul "l1".
Pentru ca am ajuns la **adancime = 0**
=> se calculeaza **V("l1") = 3** (cu functia "estimeaza_scor")
Functia alpha_beta returneaza starea "l1" cu scorul 3.
 # iesire din Apel 17 => continuare apel 16

[apel 16 continuare] Revenim in starea nodului "l" de tip **MIN**.
Se verifica daca V(fiu = "l1") poate ***minimiza*** valorile **V** si **beta** pt tata = "l".
1) Daca V("l1") < V("l"): # 3 < (+inf)
   - V("l") = V("l1") # **V("l") = 3**
   - **fiu_ales("l") = "l1"**
2) Daca V("l1") < beta("l") # 3 < (+inf)
   - beta("l") = V("l1") # **beta("l") = 3**
   - Verificam daca alpha("l") ≥ beta("l"): # 3 ≥ 3 => **True**
     => ***Retezam*** restul fiilor nodului "l" (nodul "l2").

Functia alpha_beta returneaza starea "l" cu scorul 3 si fiu_ales = "l1".
 # iesire din Apel 16 => continuare apel 13

[apel 13 continuare2] Revenim in starea nodului "j" de tip **MAX**.
Se verifica daca V(fiu = "l") poate ***maximiza*** valorile **V** si **alpha** pt tata = "j".
1) Daca V("l") > V("j"): # 3 > 2
   - V("j") = V("l") # **V("j") = 3**
   - **fiu_ales("j") = "l"**
2) Daca V("l") > alpha("j") # 3 > 3 => False

Functia alpha_beta returneaza starea "j" cu scorul 3 si fiu_ales = "l".
 # iesire din Apel 13 => continuare apel 12

[apel 12 continuare] Revenim in starea nodului "i" de tip **MIN**.
Se verifica daca V(fiu = "j") poate ***minimiza*** valorile **V** si **beta** pt tata = "i".
1) Daca V("j") < V("i"): # 3 < (+inf)
   - V("i") = V("j") # **V("i") = 3**
   - **fiu_ales("i") = "j"**
2) Daca V("j") < beta("i") # 3 < (+inf)
   - beta("i") = V("j") # **beta("i") = 3**
   - Verificam daca alpha("i") ≥ beta("i"): # 3 ≥ 3 => **True**
     => ***Retezam*** restul fiilor nodului "i" (subarborele de radacina "m").

Functia alpha_beta returneaza starea "i" cu scorul 3 si fiu_ales = "j".
 # iesire din Apel 12 => continuare apel 1

[apel 1 continuare2] Revenim in starea nodului "a" de tip **MAX**.
Se verifica daca V(fiu = "i") poate ***maximiza*** valorile **V** si **alpha** pt tata = "a".
1) Daca V("i") > V("a"): # 3 > 3 => False
2) Daca V("i") > alpha("a") # 3 > 3 => False

Functia alpha_beta returneaza starea "a" cu scorul 3 si fiu_ales = "b".

=> Concluzie algoritm:
Mutarea aleasa de calculator (jucatorul MAX) va fi cea din fiu_ales("a"), adica "b".
Aceasta mutare ii va garanta lui MAX un scor de ***cel putin 3***, indiferent de mutarile ulterioare ale lui MIN.