TRÍ TUỆ NHÂN TẠO

Bài 8: Trò chơi đối kháng không xác định

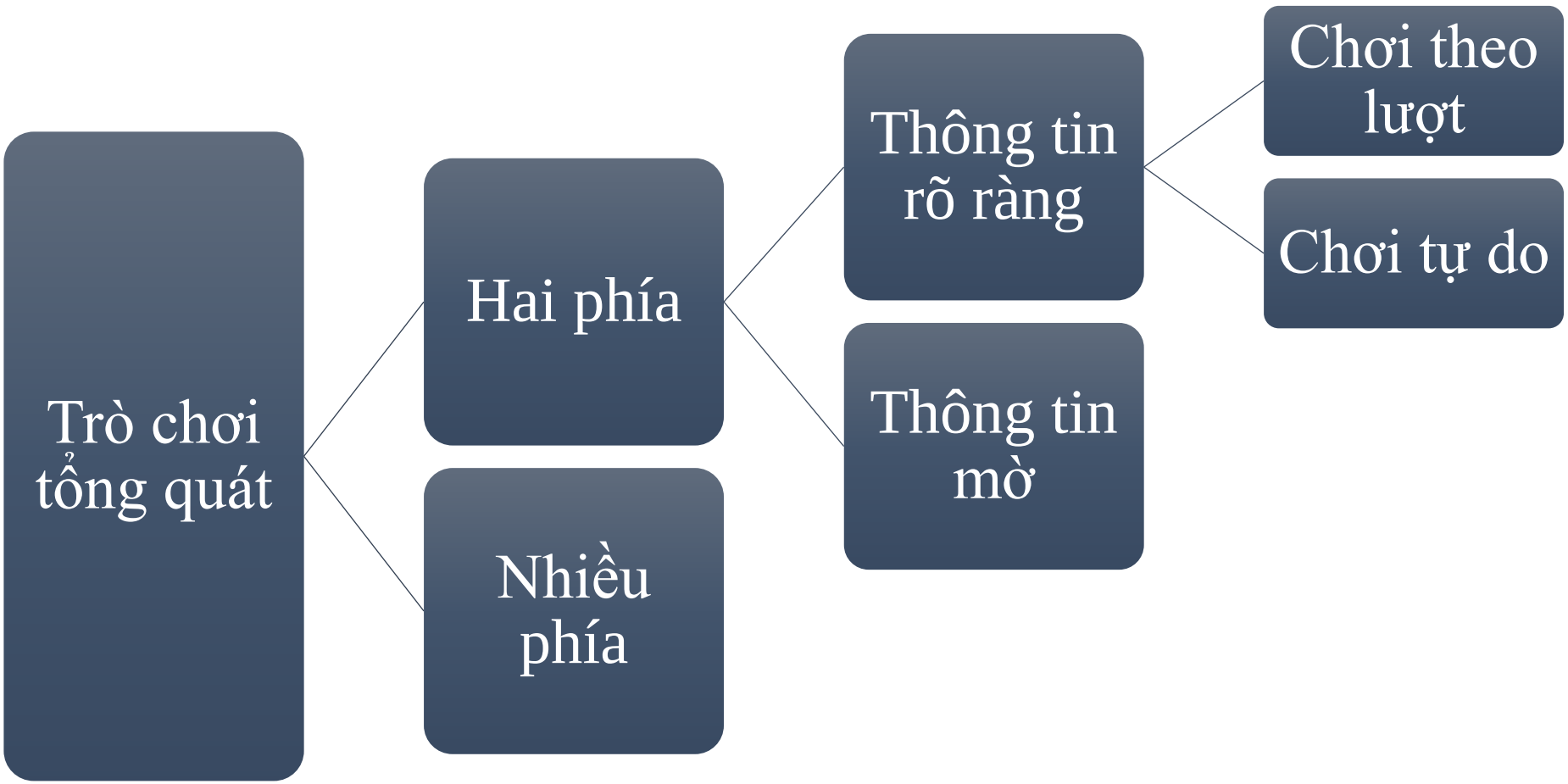
Nội dung

1. Khái niệm không xác định
2. Lượng giá Minimax
3. Thuật toán Alpha-Beta
4. Các biến thể và phát triển
5. Rủi ro và thực tế

Phần 1

Khái niệm không xác định

Phân loại trò chơi



Phân loại chiến lược chơi

Số hình trạng nhiều – KHÔNG tách được thành trò chơi con: Không tính toán được (do quá nhiều), sử dụng máy tính để tính toán các bước đi

Số hình trạng ít: Tính được trạng thái thắng-thua

Số hình trạng nhiều – tách được thành các trò chơi con: Tính trạng thái thắng thua bằng đồ thị tổng

Khái niệm không xác định

- Trò chơi đối kháng:
  - Hai người chơi
  - Quyền lợi đối lập nhau (zero-sum game)
- Trò chơi không xác định:
  - Số hình trạng quá nhiều, không thể tính toán kết cục thắng-thua
  - Không có định nghĩa rõ ràng việc thắng-thua

Phần 2

Lượng giá Minimax

Chiến lược chung

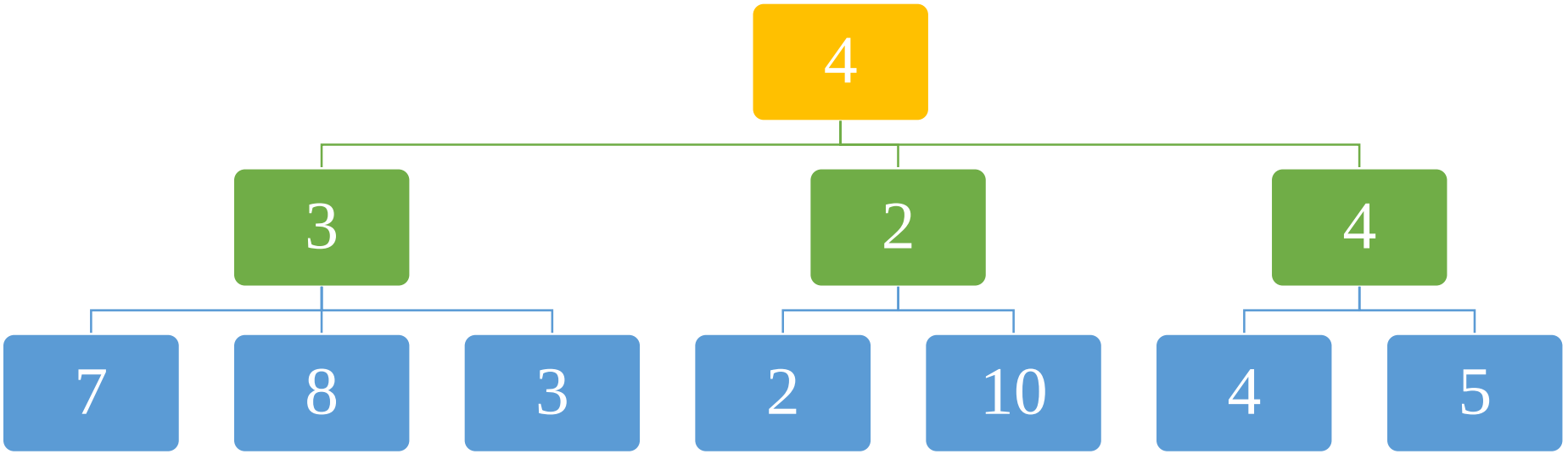
- Sử dụng công suất của máy tính mô phỏng các diễn biến có thể có của trò chơi
- Giới hạn chiều sâu để tránh bùng nổ tổ hợp
- Đưa ra một đánh giá (tương đối) cho hình trạng “cuối”
- Xây dựng chiến lược để “ép” đối phương đi vào hình trạng cuối có lợi cho máy tính

Lượng giá Minimax

- Gọi trạng thái hiện tại của trò chơi là S
- Hàm E(S) trả về số điểm đánh giá lợi thế của bên đi trước so với bên đi sau đối với S
- Diến biến trận đấu:
  - Ở lượt đầu tiên: người thứ nhất cố gắng chọn nước đi có E(S) lớn nhất (max)
  - Ở lượt thứ hai: người thứ hai cố gắng chọn nước đi để E(S) nhỏ nhất (min)
  - ….

Lượng giá Minimax

- Chiến lược chung: Tối thiểu hóa lựa chọn tốt nhất của đối phương (mini-max)



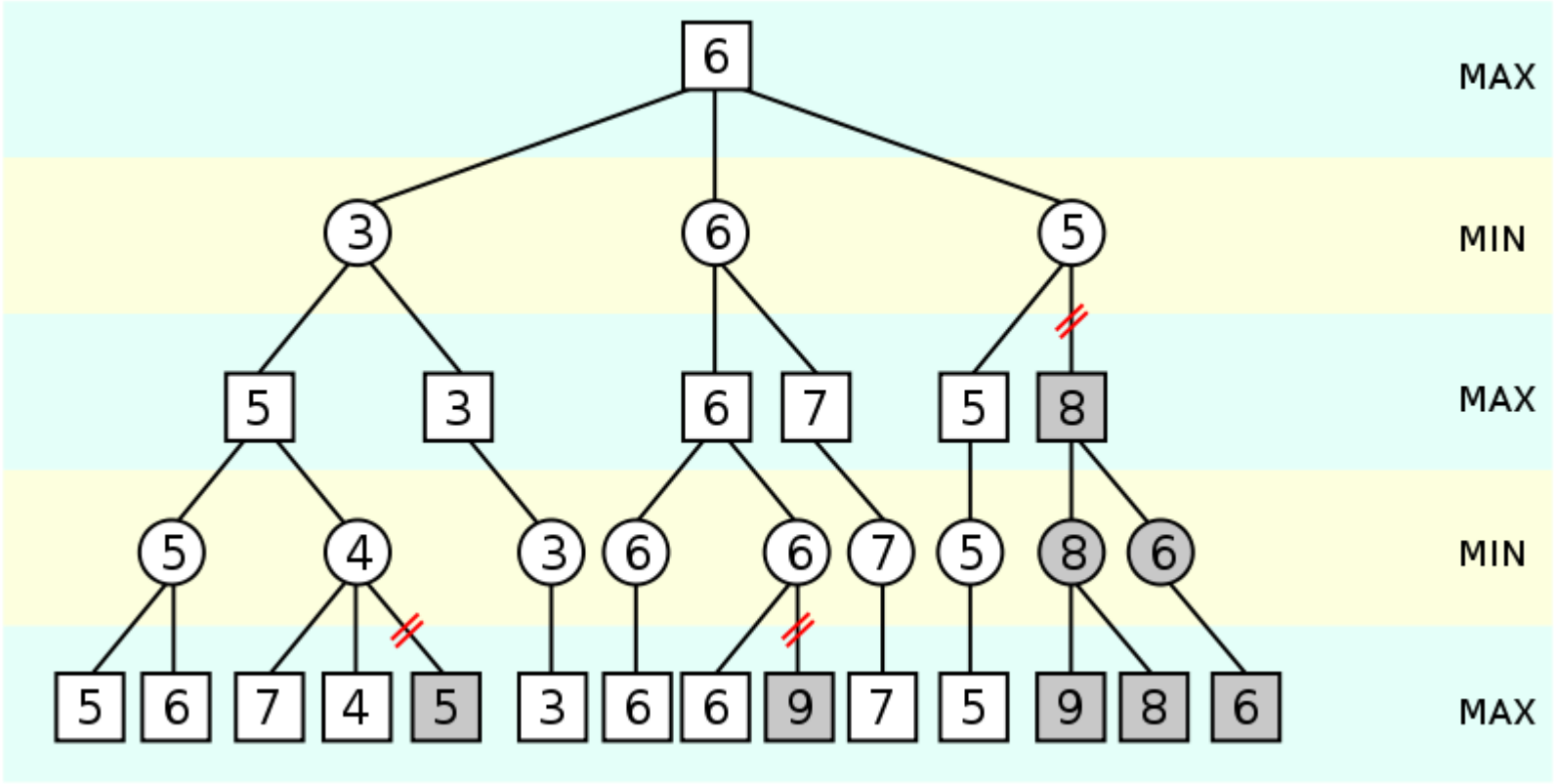
Lượng giá Minimax

```
function MAX-VALUE(state) return a value
     if  state as TERMINAL return EVALUTE(state)
     V = -∞
     for  s in  SUCCESSORS (state) do
          V = MAX(V, MIN-VALUE(s))
     return V
function MIN-VALUE(state) return a value
     if  state as TERMINAL return EVALUTE(state)
     V = +∞
     for  s in  SUCCESSORS (state) do
          V = MIN(V, MAX-VALUE(s))
     return V
function MINIMAX(state) return an action
     V = MAX-VALUE(state)
     return action ứng với giá trị V
```

Phần 3

Thuật toán Alpha-Beta

Thuật toán Alpha-Beta (1/3)



Thuật toán Alpha-Beta (2/3)

- state = Trạng thái hiện thời trong trò chơi
- α = Giá trị của giải pháp tốt nhất cho MAX dọc theo đường đi tới state
- β = Giá trị của giải pháp tốt nhất cho MIN dọc theo đường đi tới state

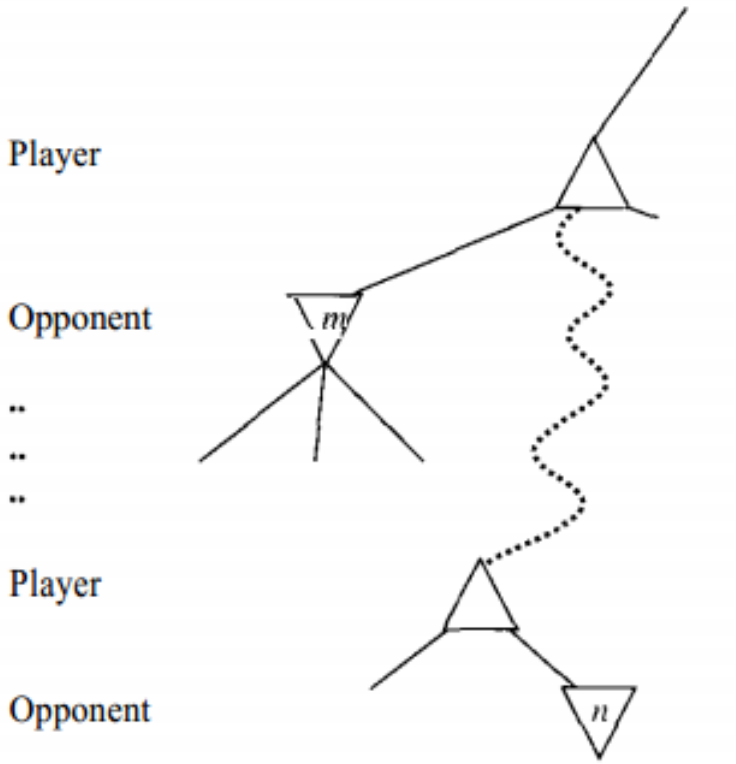
```
function MAX-VALUE(state, α, β) return a value
     if state as TERMINAL return EVALUTE (state)
     V = -∞
     for s in  SUCCESSORS (state) do
          V = MAX(V, MIN-VALUE(s, α, β))
          if V >= β return V
     α = MAX(α, V)
     return V
```

Thuật toán Alpha-Beta (2/3)

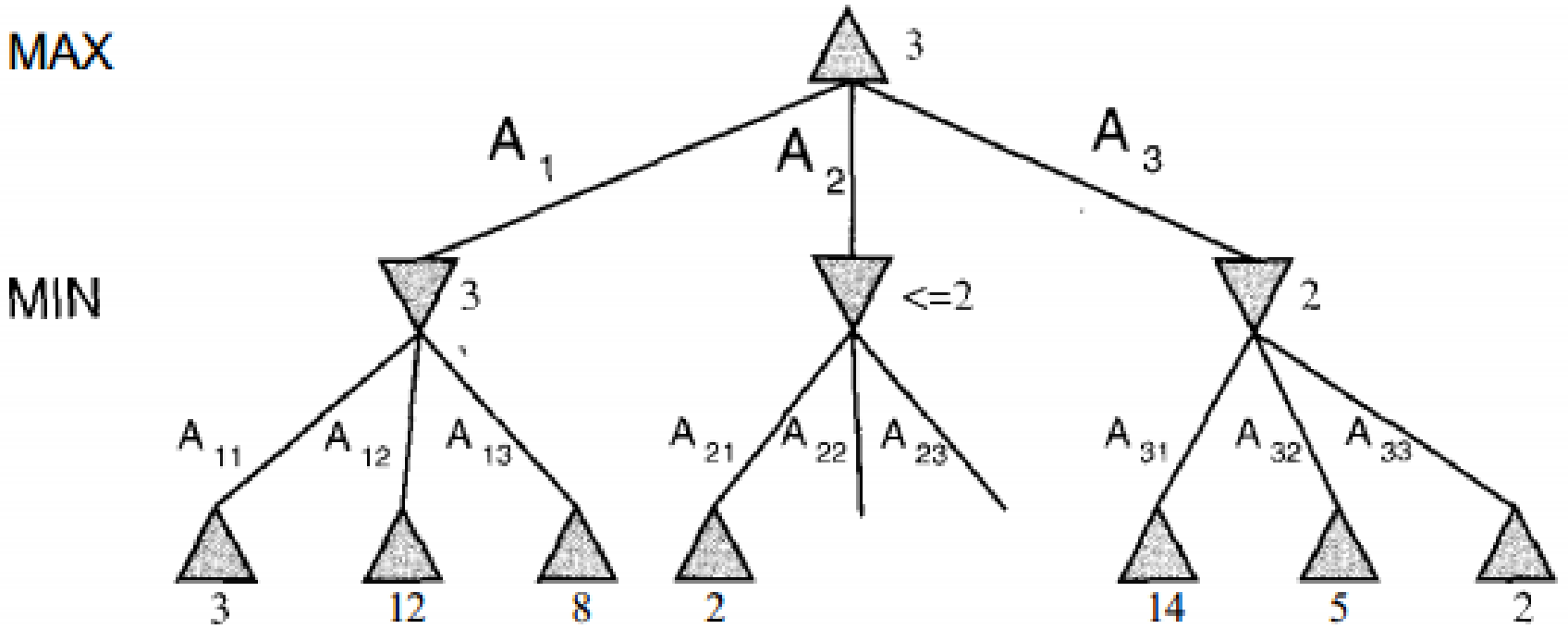
```
function MIN-VALUE(state, α, β) return a value
     if state as TERMINAL return EVALUTE (state)
     V = +∞
     for s in SUCCESSORS (state) do
          V = MIN(V, MAX-VALUE(s, α, β))
          if V <= β return V
     α = MIN(α, V)
     return V

function α-β(state) return an action
     V = MAX-VALUE(state, -∞, +∞)
     return action ứng với giá trị V
```

Hoạt động của alpha-beta

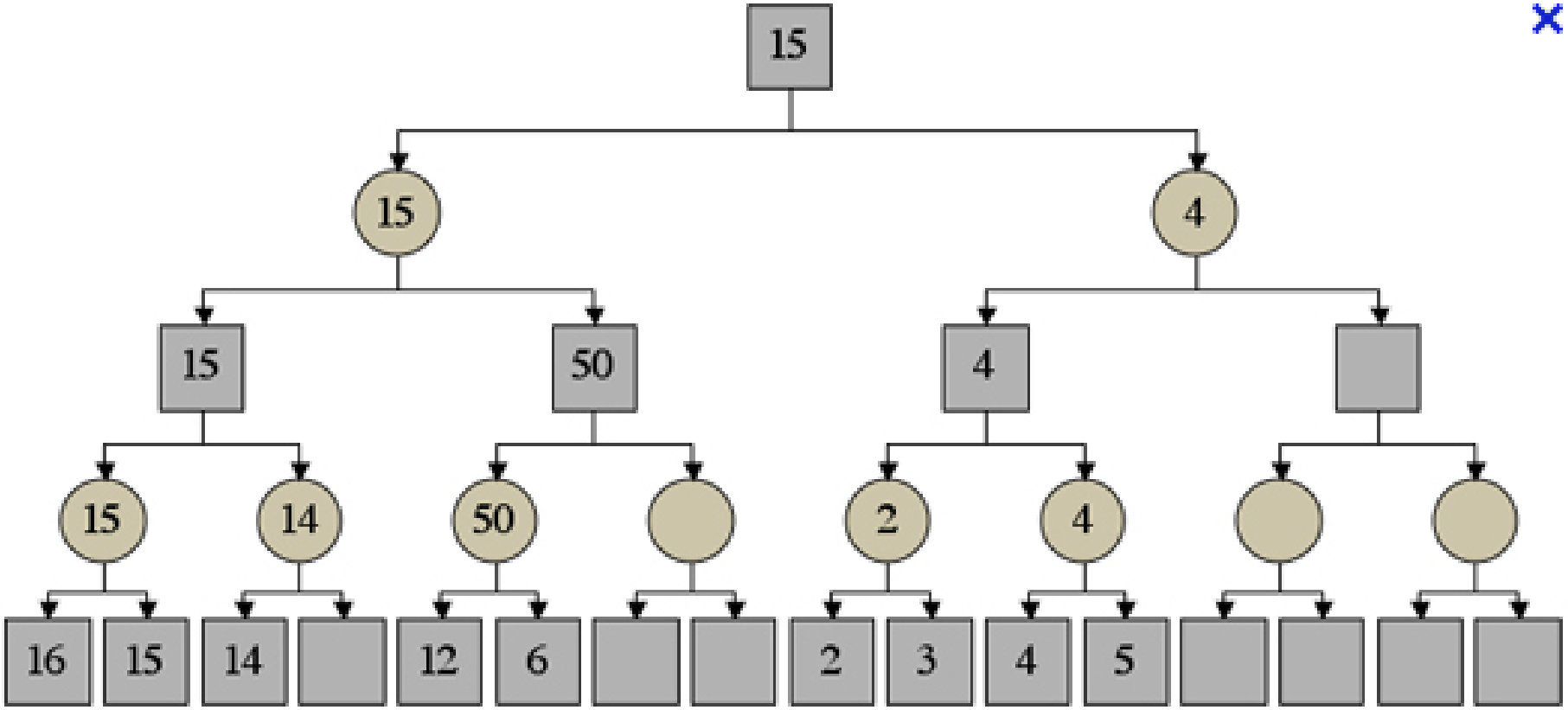
- Nguyên tắc: Khi đã tìm được nước đi m điểm
- Phía MAX: Chỉ tìm những nước đi tốt hơn (từ m+1 trở lên)
- Phía MIN: Chỉ tìm nước đi tệ hơn (từ m-1 trở xuống)



Hoạt động của alpha-beta



Hoạt động của alpha-beta



Thuật toán Alpha-Beta chuẩn

Thuật toán có thể được hiệu chỉnh ngắn gọn hơn như sau

```
function alphabeta(state, depth, α, β)
  if depth=0 return EVALUTE (state)
  if state as TERMINAL return EVALUTE (state)
  for s in SUCCESSORS (state) do
     α = max(α, -alphabeta(s, depth-1, -β, -α))
     if β ≤ α break
  return α

function α-β(state) return an action
  V = alphabeta(state, depth, -∞, +∞)
  return action ứng với giá trị V
```

Đánh giá Alpha-Beta

- Thuật toán Minimax đòi hỏi phải xét toàn bộ cây trò chơi (giới hạn độ sâu d): b^d
- Thuật toán Alpha-Beta:
  - Trường hợp tốt nhất: $b^{d/2}$
  - Trường hợp trung bình: $b^{3d/4}$
  - Trường hợp kém nhất = Minimax
- Kết quả của 2 thuật toán là như nhau

Đánh giá Alpha-Beta

- Alpha-Beta tăng trung bình được 33% độ sâu tính toán xét trên cùng một thời gian tìm kiếm
- Nếu máy tính sử dụng Minimax tính trước được 9 nửa nước đi (fly) thì dùng Alpha-Beta tính trước được 12 nửa nước đi
- Chú ý: minimax của cờ Vua ở độ sâu 9 tính toán khoảng 7.5 nghìn tỉ hình trạng

Phần 4

Các biến thể và phát triển

Các biến thể và phát triển

- Tuy đạt được hiệu quả ấn tượng như alpha-beta là chưa đủ, người ta đưa ra các ý tưởng phát triển:
  1) Tìm được nước đi tối ưu sớm nhất có thể
  2) Thăm dò và loại bỏ nước đi yếu
  3) Thu hẹp miền tìm kiếm
  4) Sử dụng lại những thông tin đã có

Các biến thể và phát triển

- NegaScout: Sử dụng cửa sổ thăm dò hẹp để thăm dò cây con
- Iterative deepening search: Tìm kiếm với độ sâu tăng dần từ 1, 2, 3,...
- Aspiration windows: Thay vì tìm kiếm trong cửa sổ đầy đủ, ta tìm trong cửa sổ hẹp hơn, VD: Độ sâu 5 ta tính được nước đi tối ưu là 120, thì ở độ sâu 6 ta chỉ cần tìm trong cửa sổ [90,150]

Các biến thể và phát triển

- Hashtable (transposition table): Lưu lại trạng thái đã tính cũ để khỏi tính lại (có lợi khi thứ tự đi quân khác nhau nhưng về cùng 1 bàn cờ)
- Late move reductions: Thu hẹp độ sâu tìm kiếm ở những ứng viên nhẹ kí
- Nullmove reductions: Thu hẹp độ sâu tìm kiếm sau khi thực hiện nước chấp

Thứ tự phát sinh nước đi

- Việc xem xét nước đi hợp lý giúp nhanh chóng tìm ra kết quả tốt, qua đó giảm thiểu số trạng thái cần tính toán
- Một vài ưu tiên khi thực hiện phát sinh nước đi (cờ Vua):
  - Main variation: nước đi tốt nhất đã biết
  - Nullmove: không đi, chấp nước đối phương
  - Killer moves: duyệt trước những nước ứng viên của best-move (thường là 2 ứng viên nặng kí nhất)
  - History heuristic: ưu tiên một số nước đi tốt của lần tìm kiếm trước. Một dạng killer-moves, lưu lại mọi nước killer-moves đã gặp (không quan trọng độ sâu)
  - Killer heuristic: nước ăn quân, thử ăn quân giá trị cao trước, ăn quân giá trị thấp sau
  - Nước đi thông thường

NegaScout

```
function negascout(state, depth, α, β)
    if depth=0 return EVALUTE(state)
    if state as TERMINAL return EVALUTE(state)
    b = β
    for s in SUCCESSORS(state) do
        α = max(α, -negascout(s, depth-1, -b, -α))
        if β ≤ α return α
        if b ≤ α
            α = -negascout(s, depth-1, -b, -α)
            if β ≤ α return α
        b = α + 1
    return α
```

Phần 5

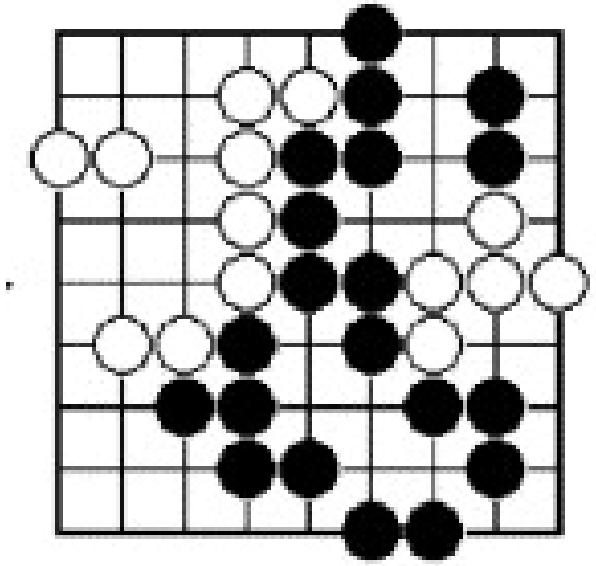
Rủi ro và thực tế

Rủi ro

- **Hiệu ứng đường giăng (horizon effect)**: Do định trước độ sâu tìm kiếm có thể máy nhìn thấy 1 hình trạng điểm cao nhưng mất quân ngay sau đó
- Cách giải quyết (phần nào) hiệu ứng đường giăng:
  - Cách tính toán giá trị "lợi thế" một cách cẩn thận
  - Quiescence search: tiếp tục tìm kiếm sâu hơn cho những hình trạng "nóng", chẳng hạn:
    - Đổi quân
    - Chiếu tướng

Chương trình chơi cờ

- Xây dựng cách đánh giá trạng thái bàn cờ (cho điểm một hình trạng)
  - Sức mạnh của các quân
  - Vị trí đứng của quân
  - Sự liên kết các nhóm quân
  - Sự an toàn của Tướng
  - …



- Đánh giá trạng thái bàn cờ cần có tri thức của chuyên gia về cờ
- Hoặc có thể sử dụng thuật toán học máy để tự tìm ra cách đánh giá phù hợp (đây chính là cách Google Alpha thực hiện)

Chương trình chơi cờ

- Tìm kiếm:
  - Alpha-Beta hoặc biến thể nào đó của nó
  - Phát sinh mọi nước đi có thể trong trạng thái hiện tại
- Cẩm nang khai cuộc: cho phép nhanh chóng chọn những nước đi đã được nghiên cứu từ trước là tốt
- Chương trình trọng tài: giúp chúng ta nhanh chóng kiểm tra xem những phiên bản cải tiến sau đó có hiệu quả hơn không