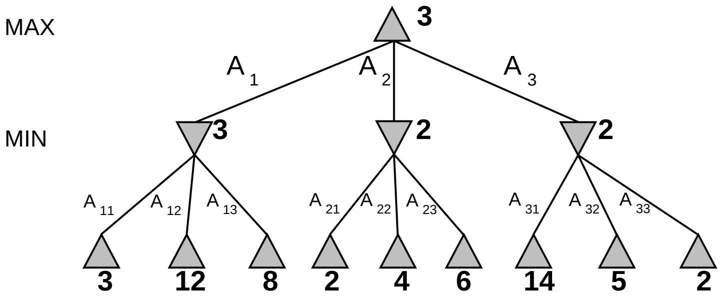
How to design AI for games

To go further

Florent Madelaine

IUT de Sénart Fontainebleau
Département Informatique

What is the best strategy?

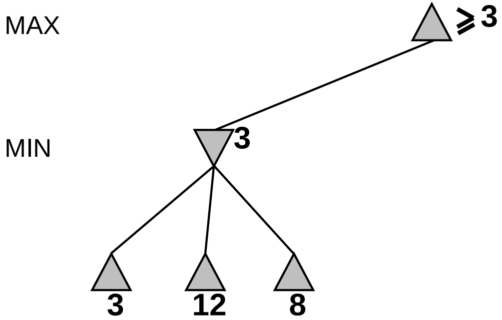


- The first player tries to maximise his payoff.
- The second player tries to minimise the first player's payoff.

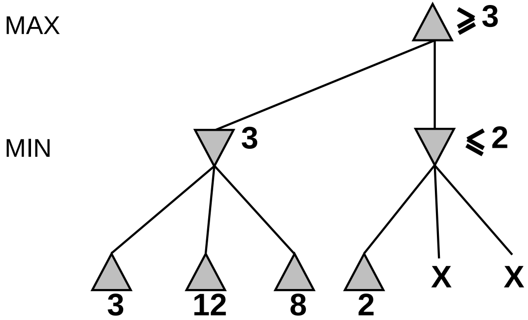
Remark

The method works also when the pay off is not just win, draw or lose.

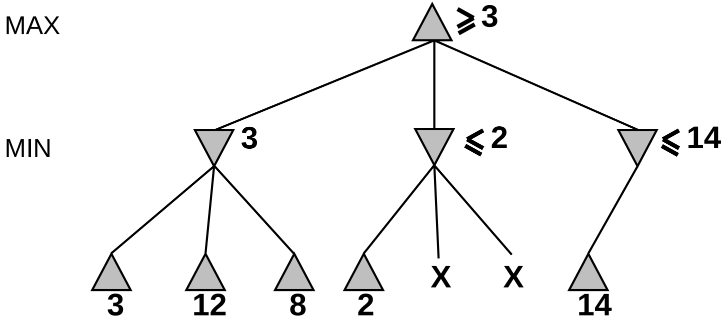
$\alpha - \beta$ pruning



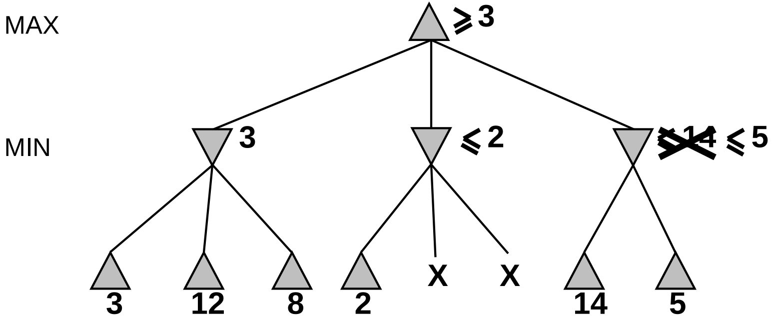
$\alpha - \beta$ pruning



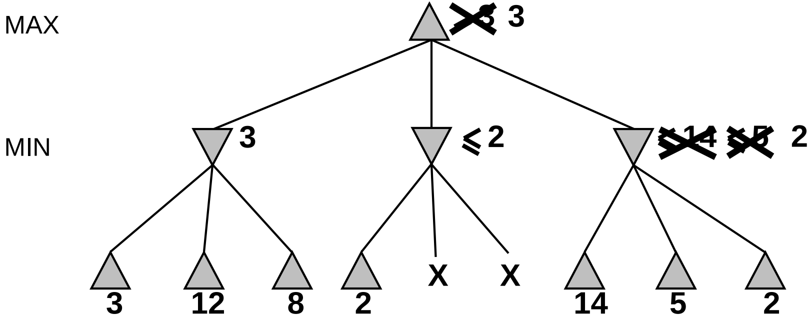
$\alpha - \beta$ pruning



$\alpha - \beta$ pruning



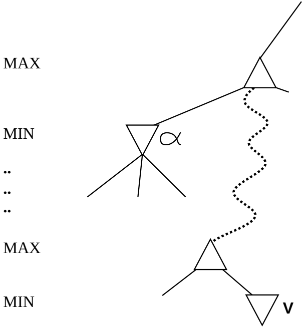
$\alpha-\beta$ pruning



Properties of $\alpha - \beta$

- Pruning *does not* affect final result.
- Good move ordering improves effectiveness of pruning.
- With “perfect ordering,” time complexity = $O(d^{h/2})$.
- This effectively *doubles* the depth of search.
- We can easily reach depth 8 and start playing good chess.

Why is it called $\alpha-\beta$?



- α is the best value (to **max**) found so far off the current path
- If V is worse than α, **max** will avoid it and prune that branch.
- Define β similarly for **min**.

From Max nodes : increasing α

ExploreMaxAlphaBeta(currentState,remainingDepth,α,β)

```
if currentState is a terminal state then
   return payoff(currentState)
else if remainingDepth = 0 then
   return eval(currentState)
else
   Localα := α// generalises bestOutcome.
   for each successor nextState do
      Outcome := ExploreMinAlphaBeta(nextState, remainingDepth−
      1, Localα, β)
      if Outcome > localα then
         localα := Outcome
         if Localα ≥ β then
            //further up in the exploration, my opponent(min) can play
            β
            // which is at least as bad for me.
         end if
      end if
   end for
   return Localα
end if
```

From Max nodes : increasing α

ExploreMaxAlphaBeta(currentState,remainingDepth,α,β)

```
if currentState is a terminal state then
    return payoff(currentState)
else if remainingDepth = 0 then
    return eval(currentState)
else
    Localα := α// generalises bestOutcome.
    for each successor nextState do
        Outcome := ExploreMinAlphaBeta(nextState, remainingDepth−
        1, Localα, β)
        if Outcome > localα then
            localα := Outcome
            if Localα ≥ β then
                //pruning step
            end if
        end if
    end for
    return Localα
end if
```

From Max nodes : increasing α

ExploreMaxAlphaBeta(currentState,remainingDepth,α,β)

```
if currentState is a terminal state then
    return payoff(currentState)
else if remainingDepth = 0 then
    return eval(currentState)
else
    Localα := α// generalises bestOutcome.
    for each successor nextState do
        Outcome := ExploreMinAlphaBeta(nextState, remainingDepth−
        1, Localα, β)
        if Outcome > localα then
            localα := Outcome
            if Localα ≥ β then
                //pruning step
                return ≥ β
            end if
        end if
    end for
    return Localα
end if
```

From Max nodes : increasing α

ExploreMaxAlphaBeta(currentState,remainingDepth,α,β)

```
if currentState is a terminal state then
   return payoff(currentState)
else if remainingDepth = 0 then
   return eval(currentState)
else
   Localα := α// generalises bestOutcome.
   for each successor nextState do
      Outcome := ExploreMinAlphaBeta(nextState, remainingDepth−1, Localα, β)
      if Outcome > localα then
         localα := Outcome
         if Localα ≥ β then
            //pruning step
            return Localα
         end if
      end if
   end for
   return Localα
end if
```

From Min nodes : decreasing β

ExploreMinAlphaBeta(currentState,remainingDepth,α,β)

```
if currentState is a terminal state then
    return payoff(currentState)
else if remainingDepth = 0 then
    return eval(currentState)
else
    Localβ := β// generalises worstOutcome
    for each successor nextState do
        Outcome := ExploreMaxAlphaBeta(nextState, remainingDepth−
        1, α, Localβ)
        if Outcome < localβ then
            localβ := Outcome
            if localβ ≤ α then
                //further up in the exploration, I (max) can play α
                // which is at least as good for me.
            end if
        end if
    end for
    return Localβ
end if
```

From Min nodes : decreasing β

ExploreMinAlphaBeta(currentState,remainingDepth,α,β)

```
if currentState is a terminal state then
    return payoff(currentState)
else if remainingDepth = 0 then
    return eval(currentState)
else
    Localβ := β// generalises worstOutcome
    for each successor nextState do
        Outcome := ExploreMaxAlphaBeta(nextState, remainingDepth−
        1, α, Localβ)
        if Outcome < localβ then
            localβ := Outcome
            if localβ ≤ α then
                //pruning step
            end if
        end if
    end for
    return Localβ
end if
```

From Min nodes : decreasing β

ExploreMinAlphaBeta(currentState,remainingDepth,α,β)

```
if currentState is a terminal state then
   return payoff(currentState)
else if remainingDepth = 0 then
   return eval(currentState)
else
   Localβ := β// generalises worstOutcome
   for each successor nextState do
      Outcome := ExploreMaxAlphaBeta(nextState, remainingDepth−
      1, α, Localβ)
      if Outcome < localβ then
         localβ := Outcome
         if localβ ≤ α then
            //pruning step
            return ≤ α
         end if
      end if
   end for
   return Localβ
end if
```

From Min nodes : decreasing β

ExploreMinAlphaBeta(currentState,remainingDepth,α,β)

```
if currentState is a terminal state then
    return payoff(currentState)
else if remainingDepth = 0 then
    return eval(currentState)
else
    Localβ := β// generalises worstOutcome
    for each successor nextState do
        Outcome := ExploreMaxAlphaBeta(nextState, remainingDepth−
        1, α, Localβ)
        if Outcome < localβ then
            localβ := Outcome
            if localβ ≤ α then
                //pruning step
                return Localβ
            end if
        end if
    end for
    return Localβ
end if
```