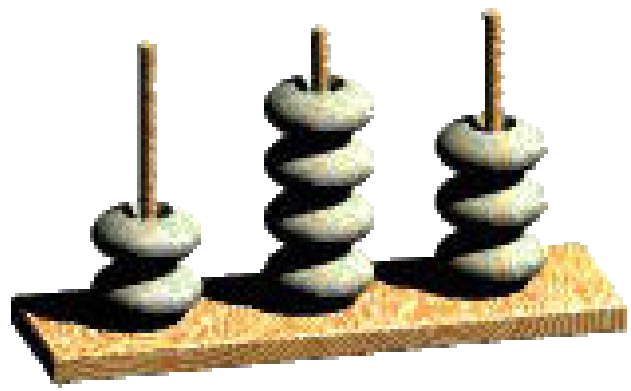
Inteligentni sistemi

Nim game, Minimax, Alpha-Beta pruning



Sta je teorija igara?

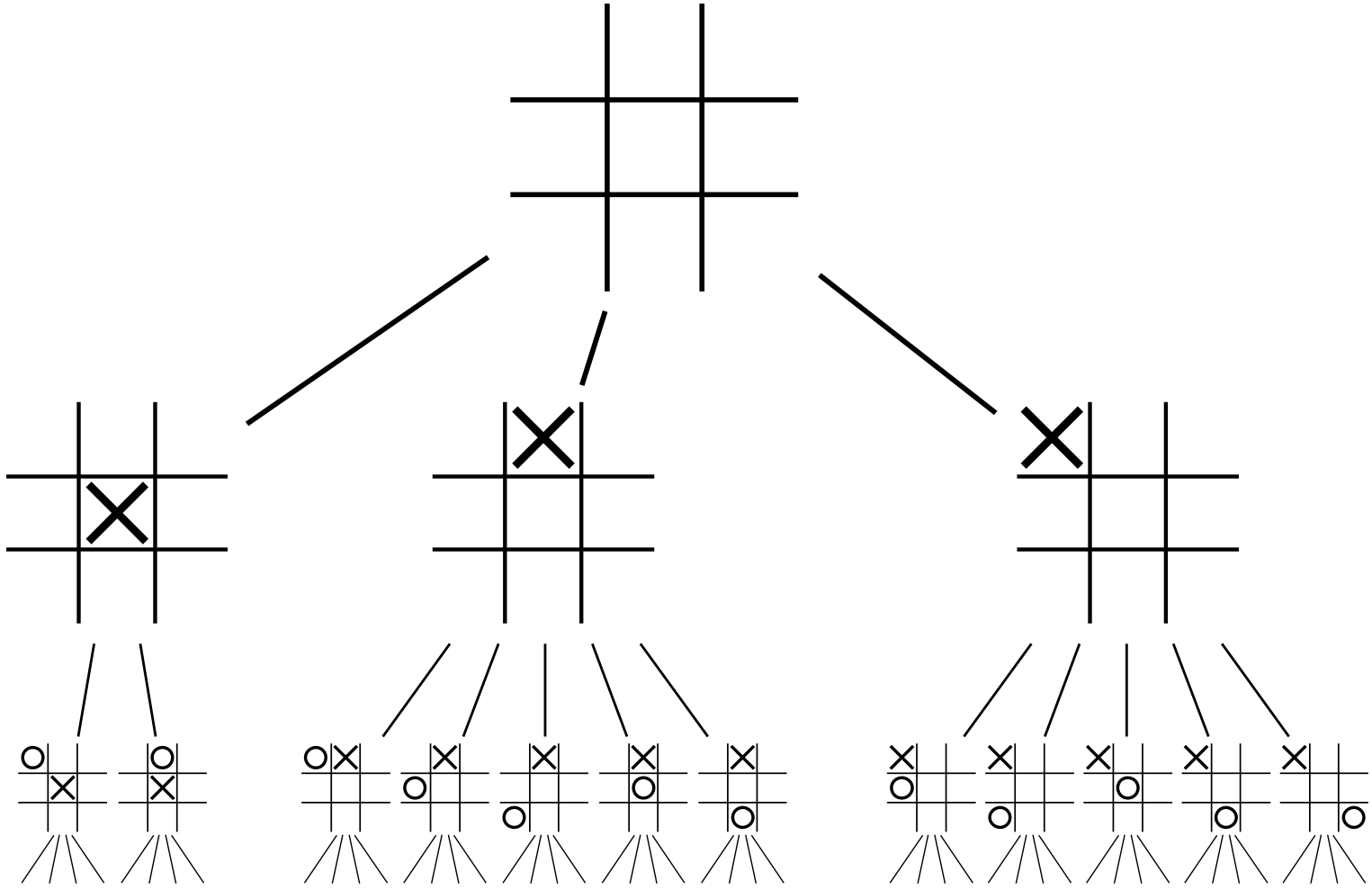
Igre mogu biti:

- kooperativne kad akteri saradjuju u zajednickom interesu i nekooperativne kada akteri pokusavaju da nadigraju jedni druge.

Upravo jedna od takvih igara je i Nim game koju cemo ovde prikazati.

Game trees

Uz pomoc njih cemo donositi odluke o narednim koracima u igri



IZGLED KLASA I OBJASNJENJE PROJEKTA NIM GAME

MainFrame.java

MainFrame klasa je klasa od koje zapocinje izvrsavanje nase aplikacije. U ovoj klasi koristimo sledece funkcije:

private void initialize() - funkcija kojom se vrsi inicijalizacija samog frame-a
I pomocu koje se podesava izgled samog frame-a.

IzborIgre.java

IzborIgre klasa, u ovoj klasi koristimo sledece funkcije:

private void initialize() - funkcija kojom se vrsi inicijalizacija samog frame-a
I pomocu koje se podesava izgled samog frame-a.

Takodje, u ovoj funkciji ce se vrsiti odabir broja stubova koji cemo koristiti u igri i odabir jednog od 3 tipa igre:

1) igrac protiv igraca
2) igrac protiv racunara
3)racunar protiv racunara

Nim.java

Nim klasa, u ovoj klasi koristimo sledece funkcije:
Ova klasa se koristi za odredjivanje stanja zetona na nasoj tabli

public Nim(Node state) - konstruktor uz pomoc koga cuvamo trenutno stanje na nasoj Nim tabeli

public Nim() - konstruktor koji ce se koristiti za kreiranje objekta Node u zavisnosti od tipa igre koji smo odabrali

private static List<Node> generateChildNodes(Node node) – funkcija pomocu koje ce za odgovarajuce prosledjeno stanje cvora (oznacenog ovde kao Node) izgenerisati lista dece (tj. sva moguca stanja u odgovarajucem Minimax stablu koje ce u korenu imati bas ovaj cvor)

private static int minimaxAlfa
(Node node, int depth, boolean maximizingPlayer, int min, int max) - Implementacija minimax algoritma sa alfa-beta odsecanjem. Izgenerise se stablo pretrazivanja i svakom od dvorova se Dodeli odgovarajuca heuristicka vrednost.

Nim.java

private static int minimax(Node node, int depth, boolean maximizingPlayer) -
Implementacija minimax algoritma sa alfa-beta odsecanjem.
Izgenerise se stablo pretrazivanja i svakom od dvorova se
Dodeli odgovarajuca heuristicka vrednost.

private static int heuristicEvaluation(Node node) – funkcija za procenu heuristike
odgovarajuceg prosledjenog cvora.

private static int nimSum(Node node) – racunanje statisticke funkcije procene

private static List<Integer> readPiles() - ispis stanja zetona na tabeli

public Node startGame() - povlacenje poteza na tabeli u zavisnosti od tipa igre

private void userMove() - prikaz poteza koji je korisnik napravio

private Node takmicarComputerMove() - koristi se kada je jedan igrac takmicarski
racunar ili kada su oba .U zavisnosti od toga se pronalazi
odgovarajuci potez koji je potrebno povuci.

Nim.java

private Node alfaComputerMove() - koristi se kada je jedan igrac alfa-beta racunar ili kada su oba .U zavisnosti od toga se pronalazi odgovarajuci potez koji je potrebno povuci.

private Node comupterMove() - koristi se kada je jedan igrac minimax racunar ili kada su oba .U zavisnosti od toga se pronalazi odgovarajuci potez koji je potrebno povuci.

private Node move() - sluzi za vracanje novog stanja na tabeli

private int readNum(String text) – za citanje broja zetona sa GUI-a

Node.java

Klasa koja se koristi za cuvanje stabla dece za odgovarajuci cvor, za cuvanje trenutnog stanja na nasoj tabeli zetona I za samu heuristicku procenu datog odgovarajuceg cvora

public Node(Integer... piles)
public Node(Node parent, List<Integer> piles - konstuktori za inicijalizaciju cvora

public List<Integer> getPiles() - dohvatanje stanja na tabeli

public List<Node> getChildList() - dohvatanje stabla pretrazivanja koje u korenu ima dati cvor. To stablo predstavlja sve moguce korake u nasoj igri.

public int getHeuristicValue() - dohvatanje heuristicke vrednosti

public void setHeuristicValue(int heuristicValue) – postavljanje heuristicke vrednosti

public boolean isEmpty() - da li je nasa tabla prazna

RasporedZetona.java

Klasa koja se koristi pri odabiru samog rasporeda zetona na nasoj tabeli.

private void initialize() - inicijalizacija samog GUI-a. Vrsi se provera da li na nasoj tabli postoje dva stuba sa istim brojem zetona.

public void addSlider(JSlider s, String description, int I) – za dodavanje komponenti Jslider-a kojima se oslikava pocetno stanje nase tabele u inicijalnom stanju.

Tezina1Racunar.java

Klasa koja se koristi u situaciji kada je jedan od igraca racunar. U njoj se odabira tezina same igre takodje bira se koji tip od moguca 3 tipa igraca ce biti nas racunar.

private void initialize() - inicijalizacija samog frame-a i unos samih podataka o tezini igre i o tipu racunara.

Tezina2Racunara.java

Klasa koja se koristi u situaciji kada su oba od igraca racunar. U njoj se odabira tezina same igre I takodje bira se koji tip od moguca 3 tipa igraca ce biti nasi racunari.

private void initialize() - inicijalizacija samog frame-a i unos samih podataka o tezini igre i o tipu svakog od oba racunara

Nim.java (nimgame)

Klasa koja ce se koristiti da se odredi koji od igraca je na potezu sledeci, a koji
Je trenutno aktivan igrac.

public Nim(int nPiles, int maxValue) – postavljanje stanja na tabeli na inicijalno stanje

public void resetGame(int nPiles, int maxValue) – funkcija koja postavlja to inicijalno stanje sistema

public void makeAMove(int index, int count) – uklanjanje odgovarajuceg broja zetona sa odgovarajuceg stuba uz pomoc funkcijie igraji zamena mesta treuntno aktivnog igraca

public void igraj(int index, int count) - funkcija za uklanjanje zetona

public abstract boolean validInput(int numberOfObjects) – provera unosa

public abstract boolean gameOver() - da li se doslo do zavrsetka igre

NimGame.java (nimgame)

Klasa koja ce se koristiti da se proveri validnost unosa i da se proveri da li je Nasa tabla prazna.

public boolean validInput(int numberOfSticks) – provera unosa

public boolean gameOver() - da li se doslo do zavrsetka igre

Pile.java (nimgame)

Klasa koja ce se koristiti da se sacuvaju podaci o broju stubova na tabeli, o broju zetona i koja pomaze pri uklanjanju zetona sa stubova.

public Pile() - postavljanje broja stubova na nultu vrednost

public Pile(int n) – odabir odgovarajuceg broja stubova

public int size() - informacija o broju stubova na tabli

public void remove(int n) – skidanje odgovarajuceg broja stubova sa table

Resenje_AlfaBeta.java (nimgame)

Klasa koja ce se koristiti u situaciji kada neki od racunara radi po principu alfa-beta odsecanja. Dakle, preduslov za koriscenje funkcija ove klase jeste da je upravo bar jedan od igraca racunar.

public Resenje_AlfaBeta() -inicijalizacija samog frame-a

private void addPiles(int nPiles) –dodavanje odabranog broja zetona na odgovarajuce stubove I povlacenje poteza u zavisnosti koji od igraca je trenutno aktivan igrac.

public int size() - informacija o broju stubova na tabli

public void remove(int n) – skidanje odgovarajuceg broja stubova sa table

public void racunarSkidaZetone(int column, int limit) – funkcija koja se poziva u situaciji kada racunar sa odredjenog stuba skida odgovarajuc broj zetona

Resenje_Takmicar.java (nimgame)

Klasa koja ce se koristiti u situaciji kada neki od racunara takmicarski racunar. Dakle, preduslov za koriscenje funkcija ove klase jeste da je upravo bar jedan od igraca racunar.

public Resenje_Takmicar() -inicijalizacija samog frame-a

private void addPiles(int nPiles) –dodavanje odabranog broja zetona na odgovarajuce stubove I povlacenje poteza u zavisnosti koji od igraca je trenutno aktivan igrac.

public int size() - informacija o broju stubova na tabli

public void remove(int n) – skidanje odgovarajuceg broja stubova sa table

public void racunarSkidaZetone(int column, int limit) – funkcija koja se poziva u situaciji kada racunar sa odredjenog stuba skida odgovarajuc broj zetona

SimpleMinimaxPlayer.java (nimgame)

Klasa koja ce se koristiti u situaciji kada neki od racunara minimaxi racunar. Dakle, preduslov za koriscenje funkcija ove klase jeste da je upravo bar jedan od igraca racunar.

public SimpleMinimaxPlayer() -inicijalizacija samog frame-a

private void addPiles(int nPiles) –dodavanje odabranog broja zetona na odgovarajuce stubove I povlacenje poteza u zavisnosti koji od igraca je trenutno aktivan igrac.

public int size() - informacija o broju stubova na tabli

public void remove(int n) – skidanje odgovarajuceg broja stubova sa table

public void racunarSkidaZetone(int column, int limit) – funkcija koja se poziva u situaciji kada racunar sa odredjenog stuba skida odgovarajuc broj zetona

Tezina2Racunara.java (nimgame)

Klasa koja ce se koristiti u situaciji kada su oba igraca racunari. Potrebno je odrediti Koji od tipova racunara su ovi igraci I shodno tome odigrati odgovarajuce poteze.

private void initialize() -inicijalizacija samog frame-a i odabranje o kom tipu igraca odnosno racunara je ovde rec

The100Game.java (nimgame)

Klasa koja ce se koristiti pri proveri da li su odgovarajuci unosi prilikom odabira tipa Igre u okvirima zadatih granica.

public boolean validInput(int numberOfSticks) – provera unosa

public boolean gameOver() - da li se doslo do zavrsetka igre

TwoComputers.java (nimgame)

Klasa koja ce se koristii u situaciji kada cemo za igrace imati dva racunara

public TwoComputers() – inicijalizacija samog frame-a postavljanje samog izgleda tabele na kojoj ce se nalaziti stubovi sa zetonima i gde ce se povlaciti potezi.

private void addPiles(int nPiles) – postavljanje izabranih zetona I samog rasporeda stubova shodno onome sto je vec u prethodnim koracima Odabrano

public void funkcija(int nPiles) – funkcija u kojoj ce se naizmenicno smenjivati povlacenje poteza dva racunara shodno vrsti racunara kojoj odgovara svaki od tih igraca.. Sami tipovi racunara su nesto sto je vec izabranao u prethodnim koracima

public void racunarSkidaZetone(int column, int limit) – uklanjanje zetona sa table za igranje

TwoPlayers.java (nimgame)

Klasa koja ce se koristii u situaciji kada za igrace nemamo racunare dve ljude. Oba nasa igraca su ljudi i oni ce povlaciti poteze na nasoj tabli sa stubovima po svojoj zelji.

public TwoPlayers()– inicijalizacija samog frame-a postavljanje samog izgleda tabele na kojoj ce se nalaziti stubovi sa zetonima i gde ce se povlaciti potezi.

private void addPiles(int nPiles) – postavljanje izabranih zetona I samog rasporeda stubova shodno onome sto je vec u prethodnim koracima odabrano

DrugiIgrac.java(pomocno)

Klasa koja ce se koristiikao pomocna pri izradi frame-a za drugog igraca

private void initialize() – inicijalizacija samog frame-a postavljanje samog izgleda tabele na kojoj ce se nalaziti stubovi sa zetonima i gde ce se povlaciti potezi

Images

Takodje, prilikom izrade samog projekta koriscene su I slicice koje se nalaze u paketima Images I img, da bi izgled samih frame-ova bio lepsi i da bi sam interfejs bio u skladu sa ocekivanjima ljubitelja igrica

Kratak prikaz implementiranog algoritma

Igra se moze shvatiti kao stablo mogucih buducih stanja. Tekuce stanje igre je koren stabla (nalazi se u vrhu). U opstem slucaju ovaj cvor ce imati nekoliko dece koja ce predstavljati sve moguce poteze koje je moguce da igrac povuce. Svako od ove dece predstavlja stanje u igri nakon sto protivnik povuce svoj potez. Ovi cvorovi ce zatim imati svoju decu… Listovi stabla su finalna stanja nase igre, tj stanja u kojima vise ni jedan potez ne moze biti povucen jer je jedan od igraca pobedio.

Minimax pretraga
Predpostavimo da dodelimo vrednost beskonacno listu koji se nalazi u stanju u kom mi pobedjujemo, a minus beskonacnost stanju u kom gubimo. Ako mozemo preci preko stabla uz pomoc ove logike moci cemo da ustanovimo da li ce trenutno aktivni igrac pobediti ukoliko bira najbolji potez.Dodeljivacemo odgovarajucu vrednost tekucem stanju u igri tako sto cemo rekurzivno koracati kroz stablo. Kad stignemo do cvorova mi cemo vratiti odgovarajucu vrednost. U cvorovima u kojima igramo, uzimamo max od vrednosti dodeljenih deci jer zelimo da odaberemo najbolju vrednost. U cvorovima u kojima se protivnik krece uzimamo min. Pogledajmo pseudo-code:

```
fun minimax(n: node): int =
   if leaf(n) then return evaluate(n)
   if n is a max node
      v := L
      for each child of n
         v' := minimax (child)
         if v' > v, v:= v'
      return v
   if n is a min node
      v := W
      for each child of n
         v' := minimax (child)
         if v' < v, v:= v'
      return v
```

Funkcija procene

Vrlo cesto je ekspandovanje citavog stabla neefikasno zato sto ima veoma veliki broj mogucih stanja. Resenje je da pretrazujemo stablo do odredjene dubine. Kada se limit dubine u pretrazi prekoraci onda se funkcija procene primenjuje na cvorove kao da se radi o cvorovima koji su listovi.

```
(* the minimax value of n, searched to depth d *)
 fun minimax(n: node, d: int): int =
   if leaf(n) or depth=0 return evaluate(n)
   if n is a max node
      v := L
      for each child of n
         v' := minimax (child,d-1)
         if v' > v, v:= v'
      return v
   if n is a min node
      v := W
      for each child of n
         v' := minimax (child,d-1)
         if v' < v, v:= v'
      return v
```

Alfa – beta pruning

Minimax pretrazuje neke delove stabla koje zapravo I ne mora. Izbegavanje Pretrage delova stabla se naziva odsecanje, I mi cemo koristiti alfa-beta od Odsecanje. Koristicemo granica sa nazivima min I max da odsecemo delove stabla(tj podstabla) tako sto cemo da prekinemo pretragu stabla ranije.

Pogledajmo sledeci pseudo-code:

```
(* the minimax value of n, searched to depth d.
 * If the value is less than min, returns min.
 * If greater than max, returns max. *)
 fun minimax(n: node, d: int, min: int, max: int):
int =
   if leaf(n) or depth=0 return evaluate(n)
   if n is a max node
      v := min
      for each child of n
         v' := minimax (child,d-1,...,...)
         if v' > v, v:= v'
         if v > max return max
      return v
   if n is a min node
      v := max
      for each child of n
         v' := minimax (child,d-1,...,...)
         if v' < v, v:= v'
         if v < min return min
      return v
```

```
(* the minimax value of n, searched to depth d.
 * If the value is less than min, returns min.
 * If greater than max, returns max. *)
 fun minimax(n: node, d: int, min: int, max: int): int =
   if leaf(n) or depth=0 return evaluate(n)
   if n is a max node
      v := min
      for each child of n
         v' := minimax (child,d-1,v,max)
         if v' > v, v:= v'
         if v > max return max
      return v
   if n is a min node
      v := max
      for each child of n
         v' := minimax (child,d-1,min,v)
         if v' < v, v:= v'
         if v < min return min
      return v
```

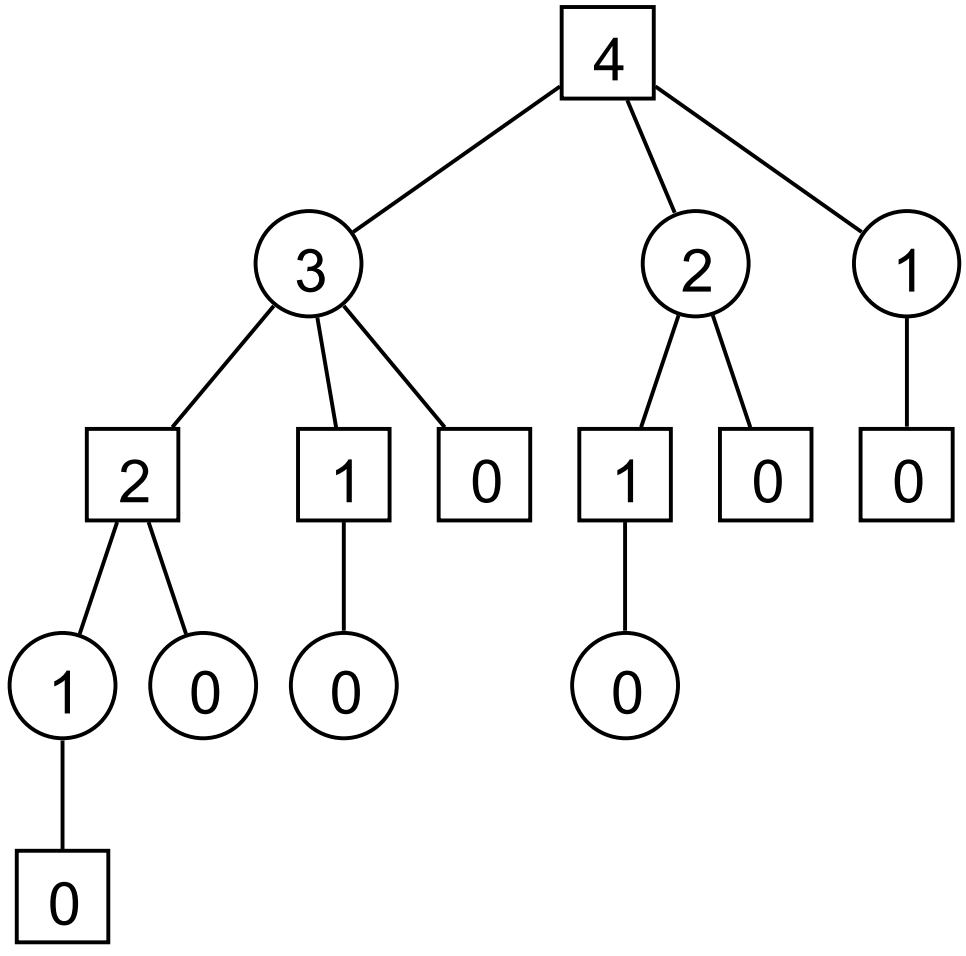
Takmicarski igrac

Koliko je efektivno alfa-beta odsecanje? To zavisi od redosleda u kom se deca obilaze. Ako se deca cvora obilaze u u najgorem moguce redosledu, onda se mozda odsecanje nece ni dogoditi. Za max cvorove mi zelimo da posetimo najbolje dete najpre tako da ne trosimo vreme na ostalu decu. Za min cvorove mi zelimo da pretrazimo najgore dete prvo(iz nase perspektive, ne protivnikove).

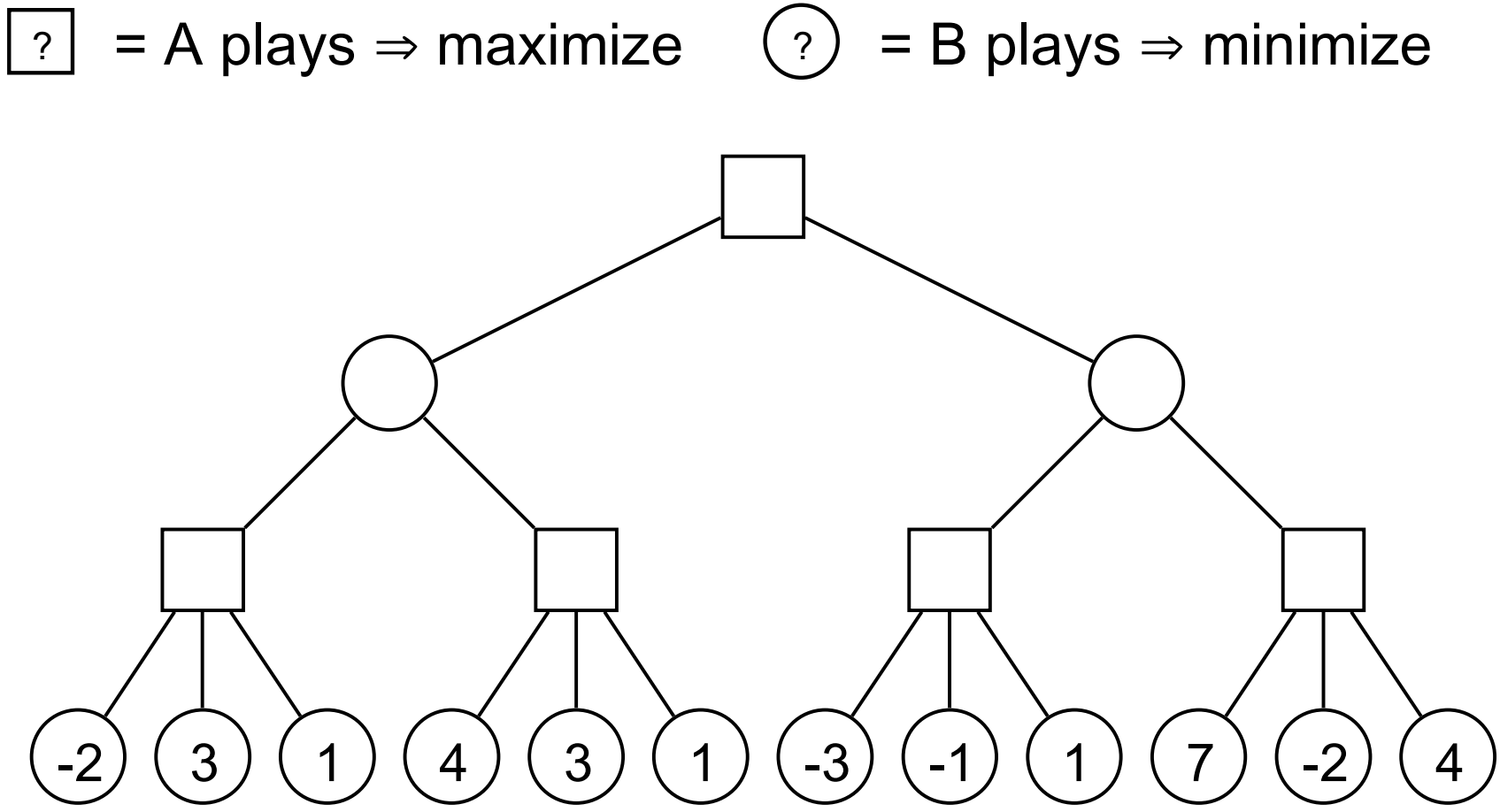
Ovde kod nas smo izabrali da funkcija procene se koristi za rangiranje dece cvorova. Kada je optimalno dete izabrano onda ce alfa-beta odsecanje dovesti do toga da se sva ostala deca odseku na svakom drugom nivou stabla, samo taj cvor ce biti pretrazen. To znaci da se moze postici veliki napredak u performansama

Bachet's game: game tree

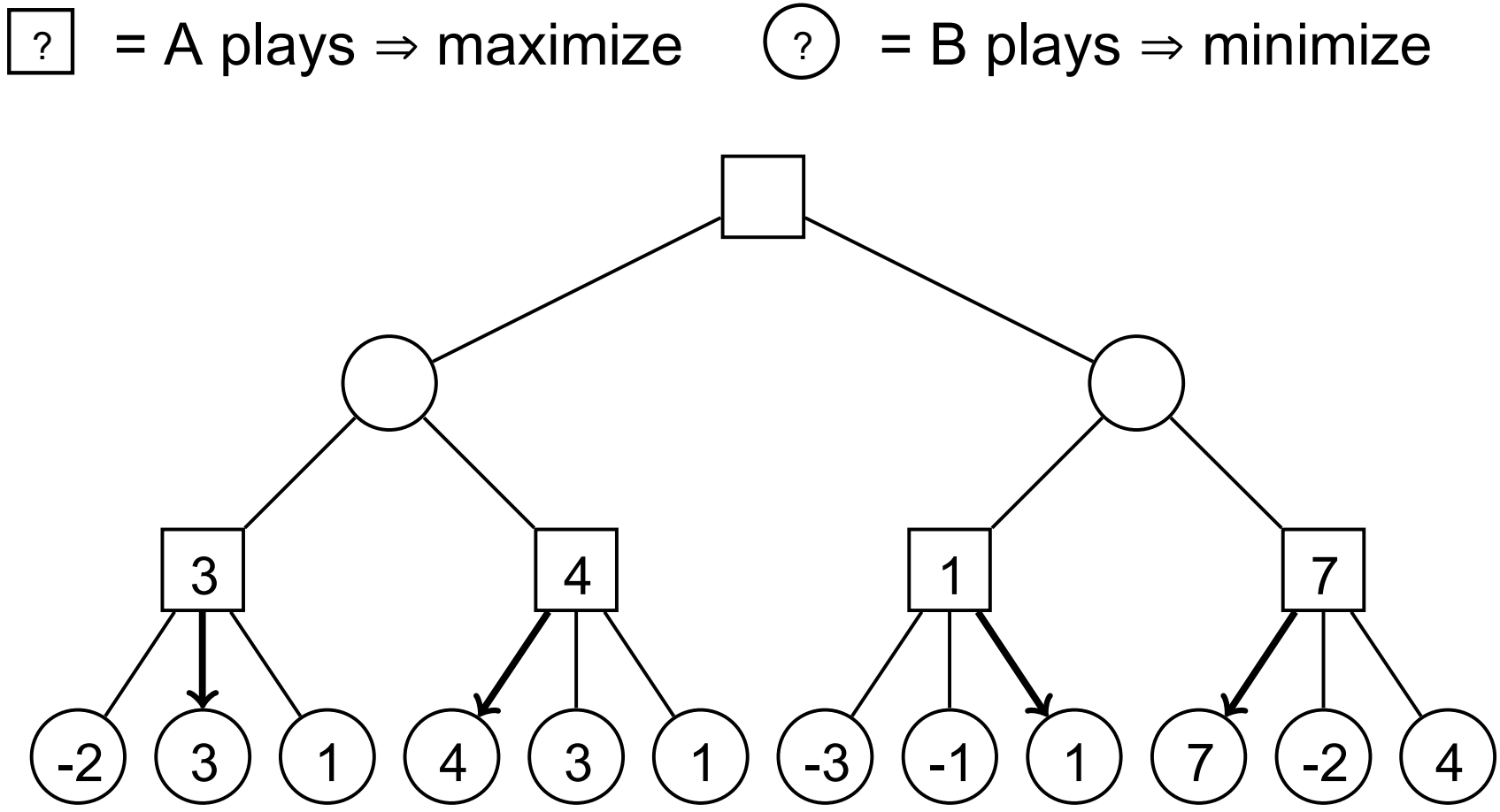
? = A plays ? = B plays



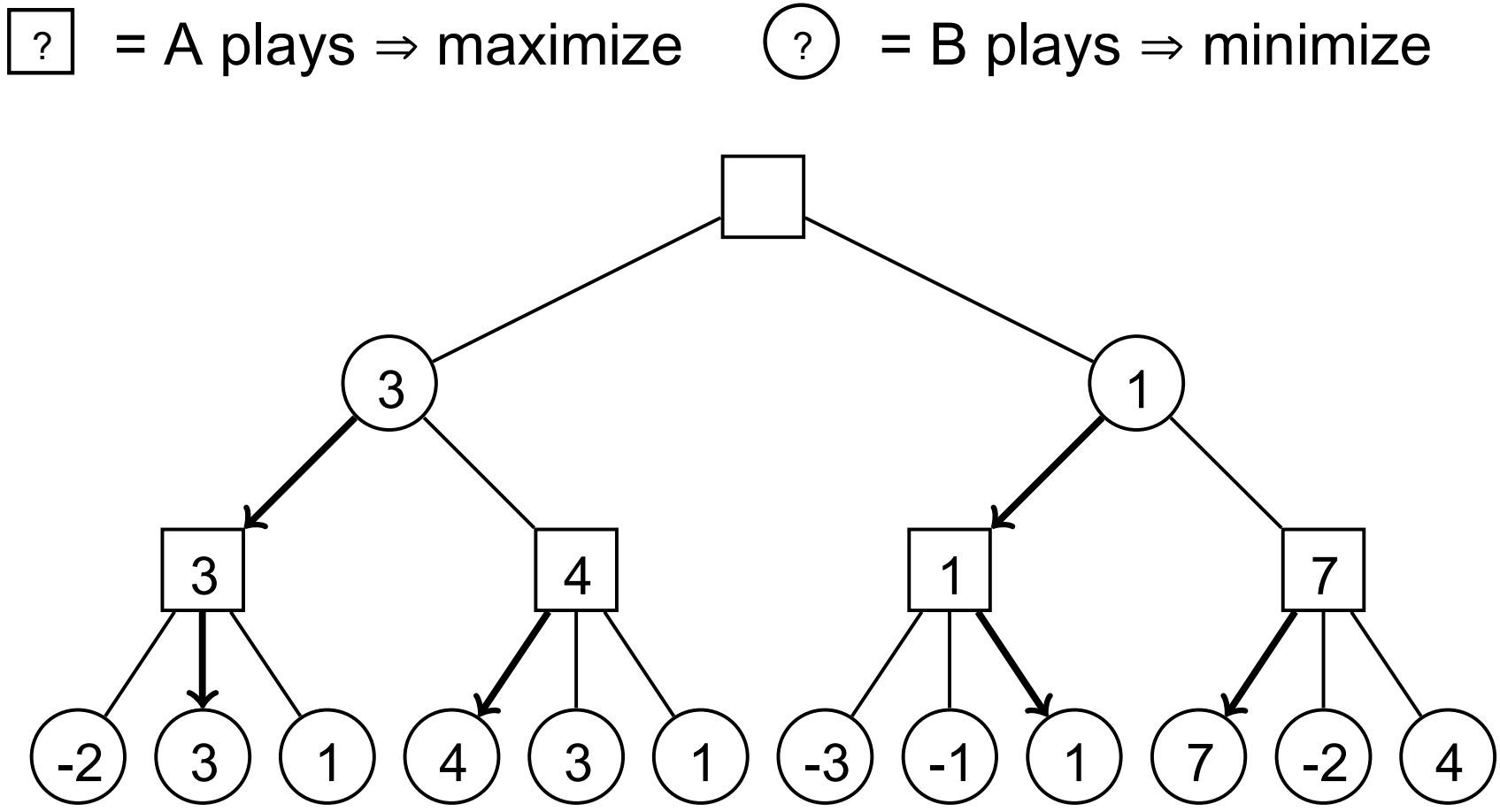
Minimax example



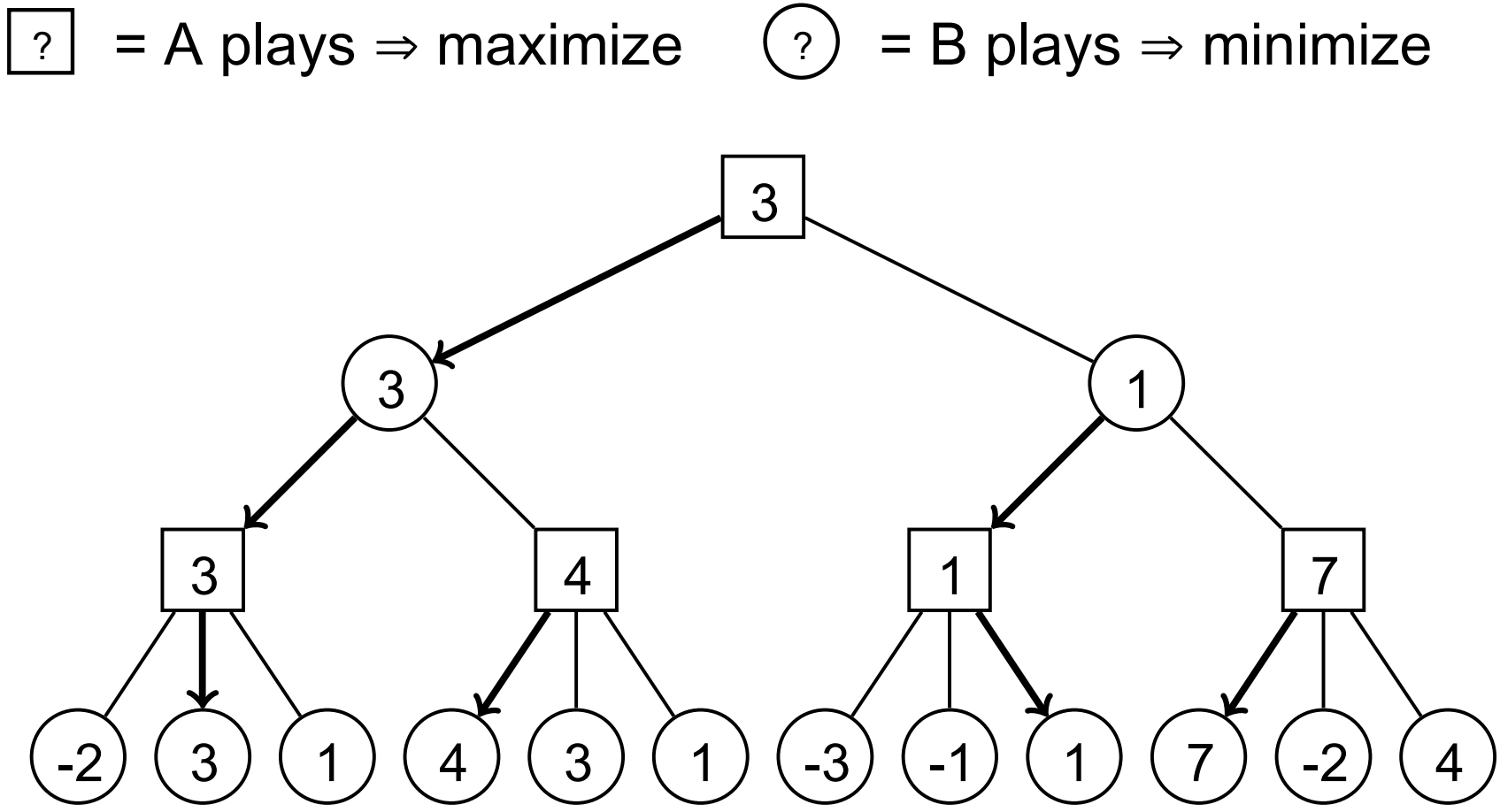
Minimax example



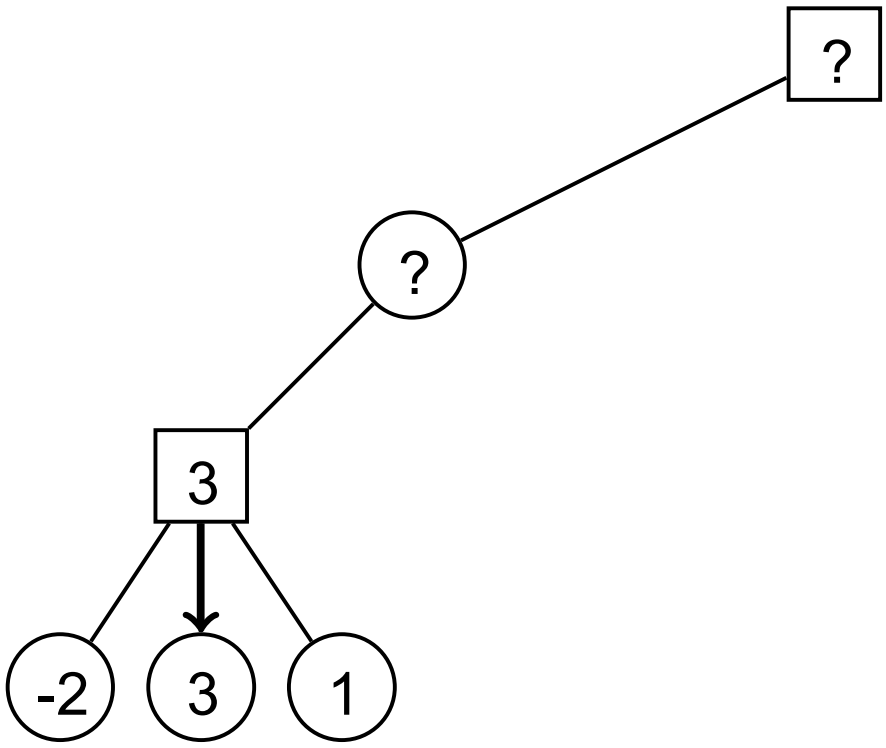
Minimax example



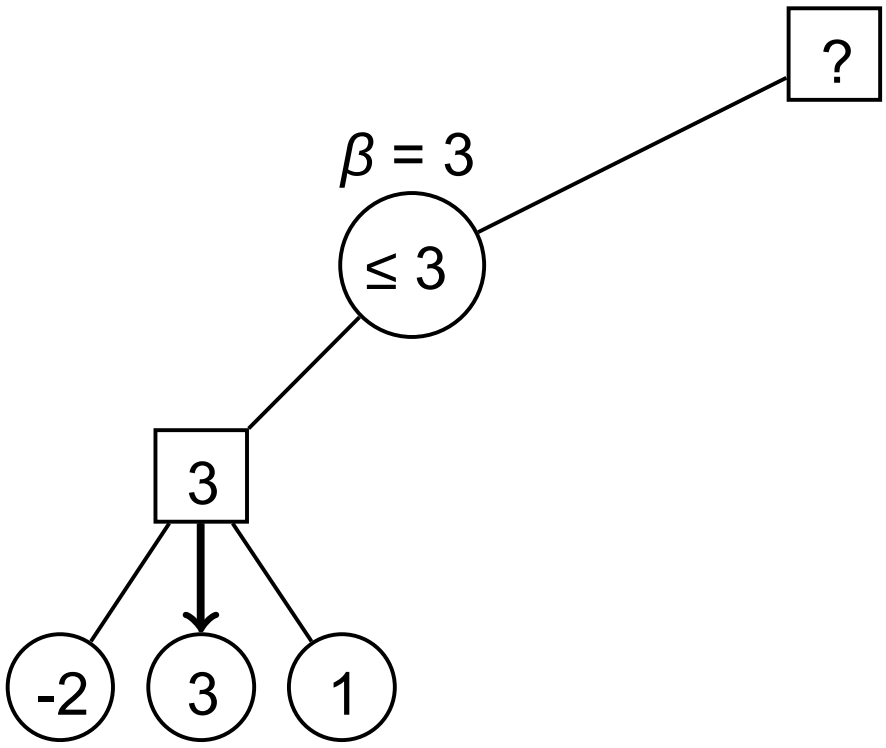
Minimax example



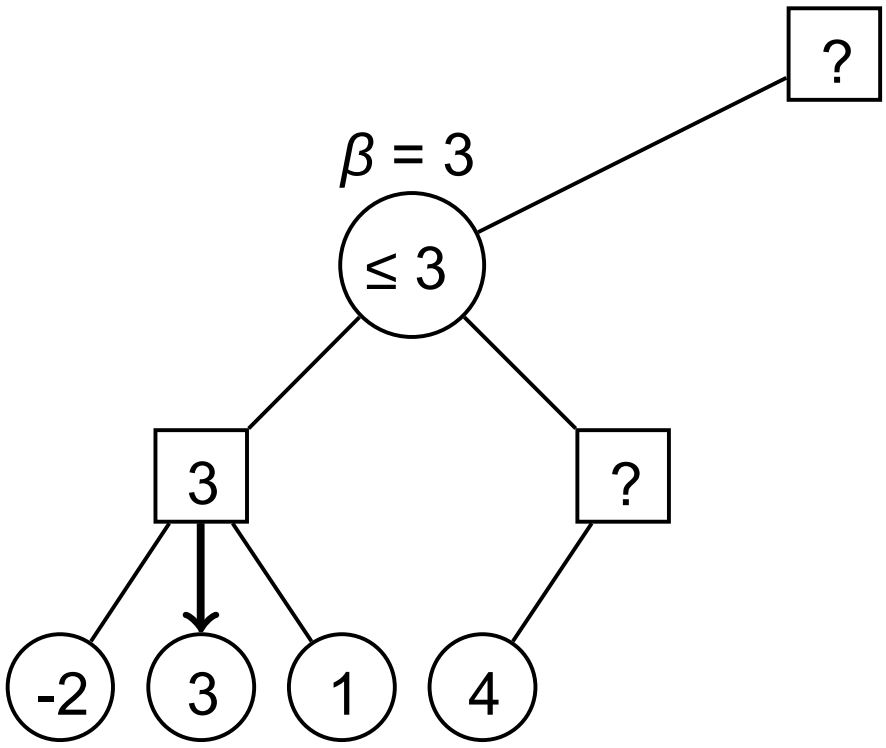
Alpha-beta example



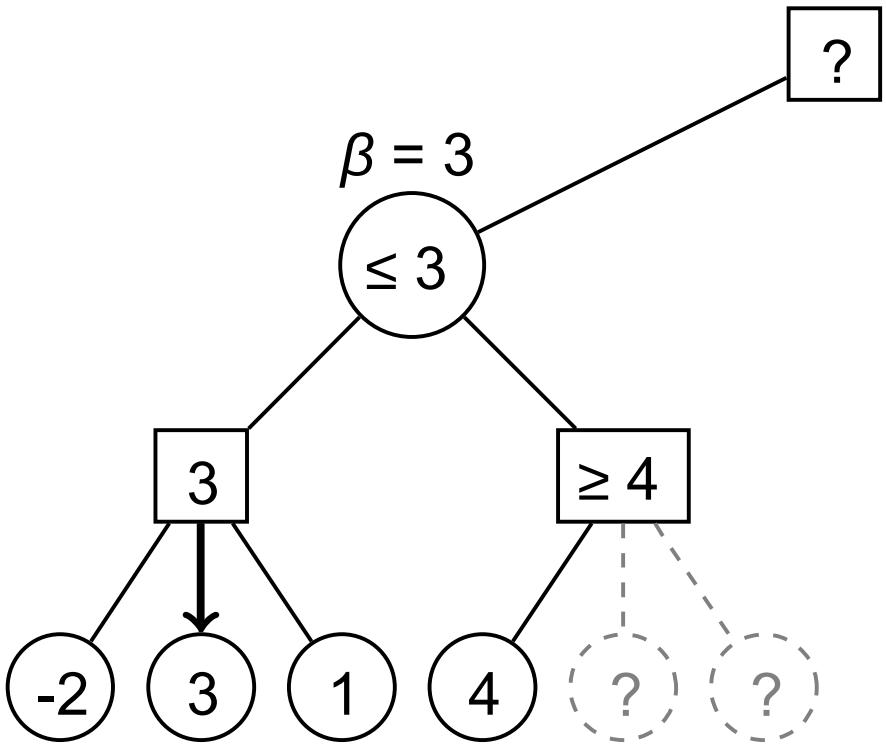
Alpha-beta example



Alpha-beta example

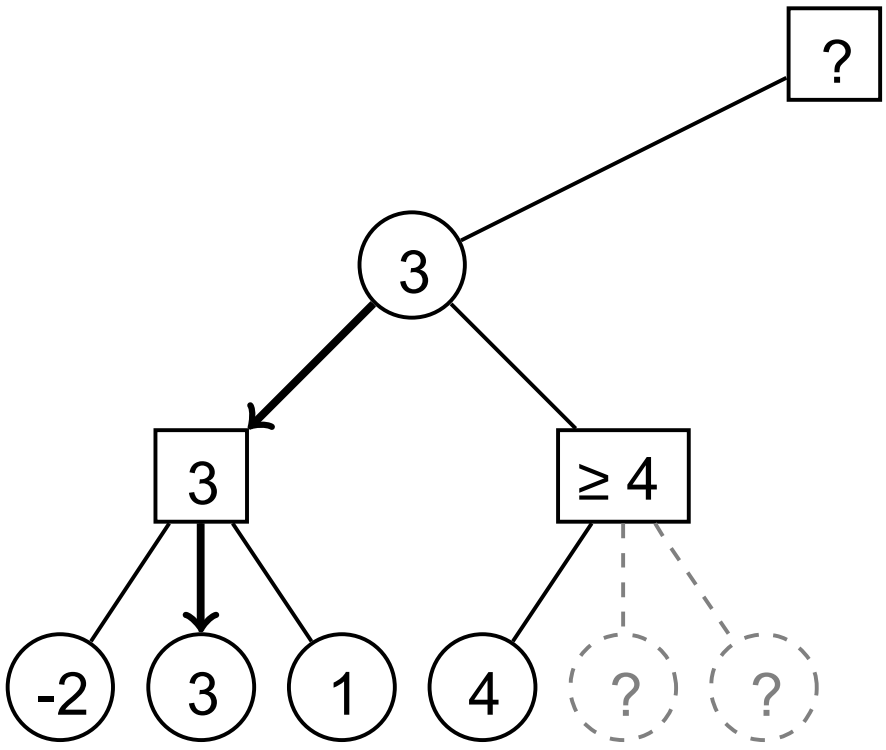


Alpha-beta example



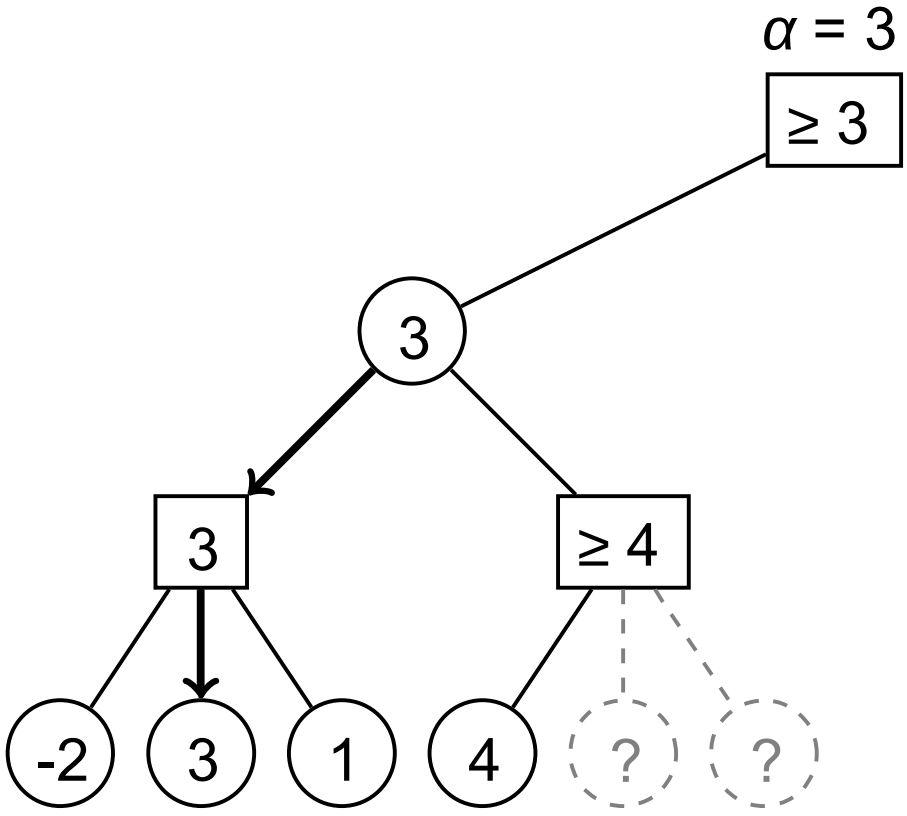
beta pruning!

Alpha-beta example



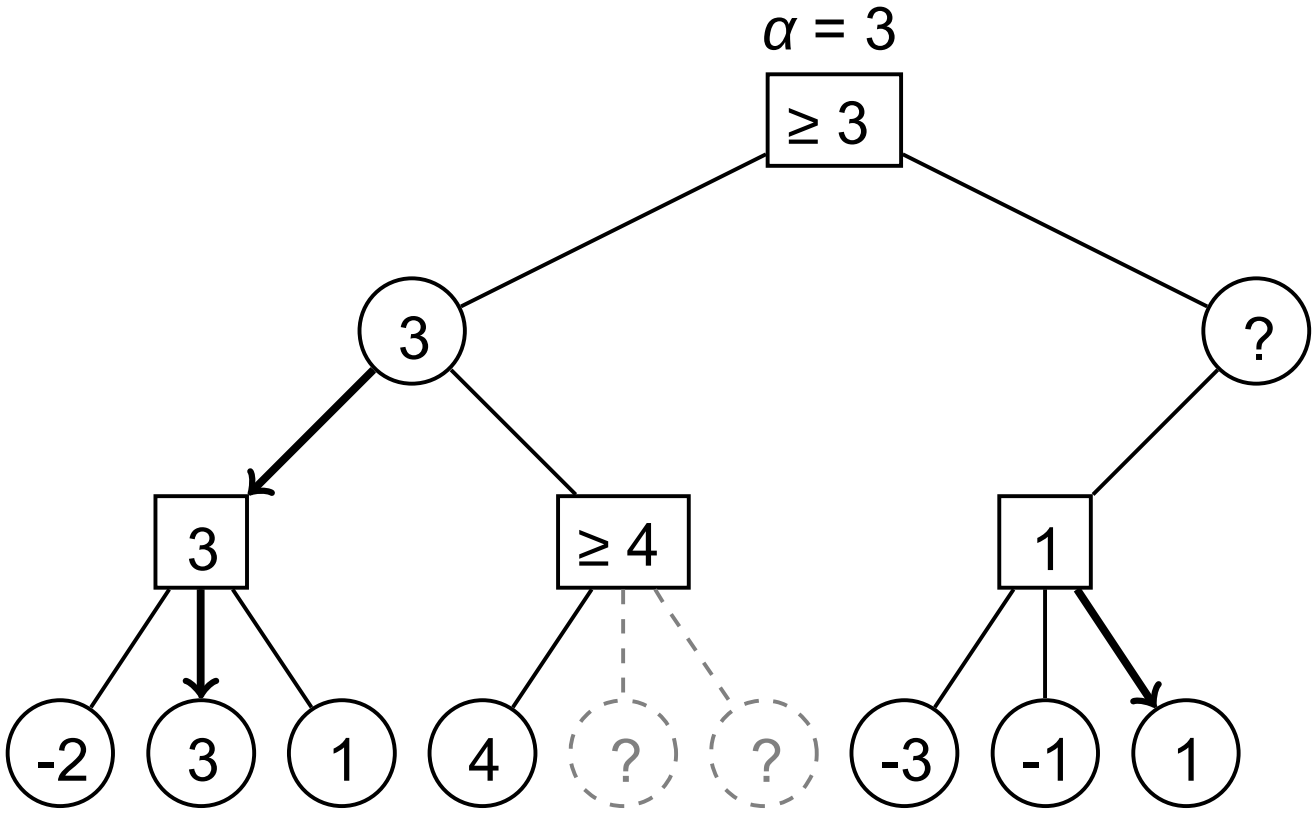
beta pruning!

Alpha-beta example



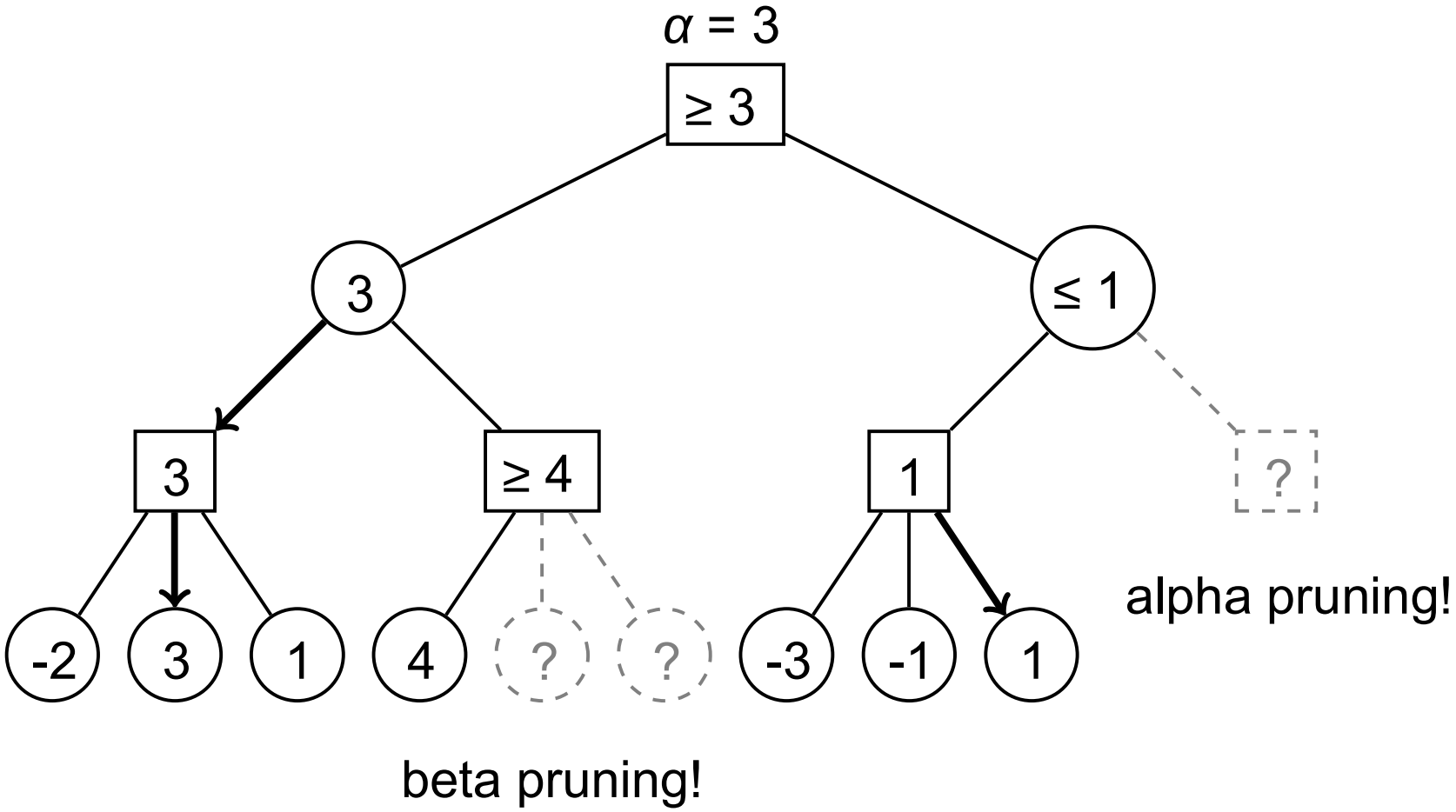
beta pruning!

Alpha-beta example



beta pruning!

Alpha-beta example



Alpha-beta example