- Aiemmin etsittiin tekojonoa alkutilasta maalitilaan. Nyt tilanne vaikeutuu, sillä tarkasteluun otetaan muut agentit, jotka voivat pyrkiä estämään tavoitteeseen pääsyä.
- Yleisiä vuorovaikutustilanteita käsitellään *peliteoriassa* (game theory). Nyt keskitytään kahden agentin peleihin, joissa kaikki tieto ympäristöstä on saatavilla ja joissa vallitsee tiukka kilpailuasetelma, siis nk. *täydellisen informaation nollasummapeleihin* (zero-sum games of perfect information).
- Pelit ovat olleet suosittuja tutkimuskohteita toisaalta siksi että ne ovat haastavia ja toisaalta siksi että algoritmien toimintaa on helppo verrata ihmisen toimintaan.
- Peleissä koko etsintäavaruuden läpikäyminen on yleensä mahdotonta joten on kokeiltava siirtoja vaikka niiden optimaalisuudesta ei ole varmuutta. Yleensä tarvitaankin heuristisia evaluointifunktioita, joiden avulla arvioidaan pelitilanteiden hyvyyttä.

OPTIMAALISET PÄÄTÖKSET PELEISSÄ

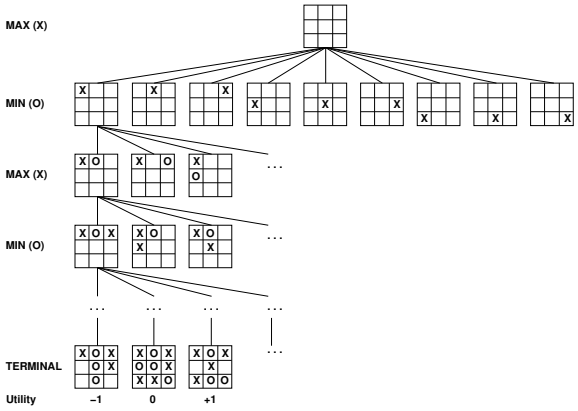
- Oletetaan kaksi pelaajaa: MAX ja MIN. Pelin aloittaa MAX ja sen jälkeen pelaajat tekevät siirtonsa vuorotellen.
- Peli voidaan määritellä etsintäongelmana:

  Alkutila: Pelin (pelilaudan) alkutilanne ja aloittavan pelaajan nimi.

  Seuraajafunktio: Palauttaa pelin sääntöihin perustuen kustakin tilasta teko-tila-parin, joka kertoo mihin tilaan päädytään milläkin teolla.

  Maalitesti: Määrittää pelin lopputilat eli tilanteet, joissa peli päättyy.

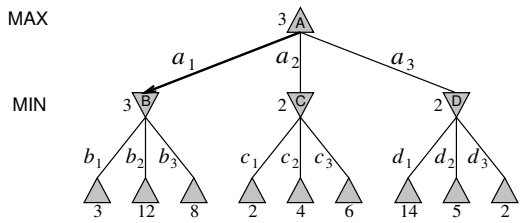
  Polkukustannus: Utiliteettifunktio, joka sijoittaa kullekin lopputilalle numeroarvon, esim. voittotiloille +1, tappiotiloille -1 ja tasapelitiloille 0.
- Näiden perusteella saadaan pelin *pelipuu* (game tree), jossa pelaajat vuorotellen valitsevat kuljettavan kaaren.



STRATEGIAT

- Pelkkä teko- eli siirtojonon löytäminen ei peleissä onnistu, koska toinen pelaaja muuttaa omilla teoillaan ympäristön tilannetta. Pelaajien tehtävänä onkin löytää *strategiat*, jotka määrittävät tehtävät siirrot alkutilassa ja kussakin mahdollisessa vastustajan siirrosta syntyneessä tilanteessa.
- Optimaalinen strategia on sellainen, joka päätyy aina vähintään yhtä hyviin lopputiloihin kuin mikä tahansa muu strategia kun oletetaan että vastustaja on erehtymätön ja siis käyttää omaa optimaalista strategiaansa.
- Optimaalisen strategian määrittämiseksi kullekin solmulle lasketaan *minimax-arvo* seuraavasti:

$$MM(n) = \begin{cases} \text{UTILITY}(n) & \text{jos } n \text{ on loppusolmu} \\ \max_{s \in Succ(n)} MM(s) & \text{jos } n \text{ on MAX-solmu} \\ \min_{s \in Succ(n)} MM(s) & \text{jos } n \text{ on MIN-solmu} \end{cases}$$



- Kahden optimaalista strategiaa käyttävän pelaajan pelinkulku määräytyy täysin minimax-arvojen mukaan.
- Jos toinen pelaaja, vaikkapa MIN ei pelaa optimaalisesti, niin MAX pärjää vähintään yhtä hyvin kuin optimaalista pelaajaa vastaan. (Jokin muu strategia kuin minimax saattaa kuitenkin tällöin tuottaa vielä paremman tuloksen.)

MINIMAX-ALGORITMI

function MINIMAX-DECISION(*state*) **returns** *an action*
  inputs: *state*, current state in game

  return the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

function MAX-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do** $v \leftarrow$ MAX(v, MIN-VALUE(*s*))
  return *v*

function MIN-VALUE(*state*) **returns** *a utility value*
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow \infty$
  for *a*, *s* in SUCCESSORS(*state*) **do** $v \leftarrow$ MIN(v, MAX-VALUE(*s*))
  return *v*

- Näin esitetty algoritmi käy pelipuun lävitse syvyyssuuntaisesti ja on täydellinen jos puu on äärellinen.
- Algoritmi on optimaalinen, jos vastustaja pelaa optimaalisesti.
- Algoritmin aikavaatimus on luokkaa $O(b^m)$, missä b on sallittujen siirtojen maksimimäärä ja m on pelipuun maksimisyvyys.
- Tilavaatimus on luokkaa $O(bm)$, jos kaikki seuraajat generoidaan kerralla ja $O(m)$ jos ne generoidaan yksi kerrallaan kuten peruuttavassa etsinnässä.
- Eksponentiaalinen aika on liikaa oikeissa pelipuissa. Esim. shakissa $b \approx 35, m \approx 100$.

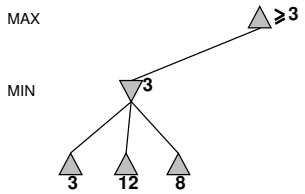
ALFA–BEETA-KARSINTA

- Minimax-strategian laskemista voidaan tehostaa, koska kaikkia pelipuun haaroja ei tarvitse tutkia.
- Esim. edellisen puun minimax-arvoa laskettaessa kaksi lehteä voidaan jättää tutkimatta, koska

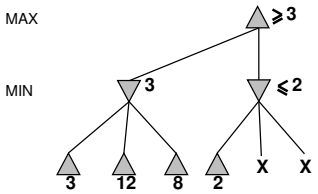
$$\begin{aligned} MM(\text{juuri}) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\ &= \max(3, \min(2,x,y), 2) \\ &= \max(3,z,2), \text{ missä } z \leq 2 \\ &= 3. \end{aligned}$$

  Juuren minimax-arvo ei siis riipu lehtien x ja y arvoista.
- Siis kun karsinnassa tarkastellaan solmuun n siirtymistä, niin jos valitsevalla pelaajalla on jo parempi valintavaihtoehto löydettynä jossakin vanhempisolmussa, ei solmua n koskaan valita pelissä.

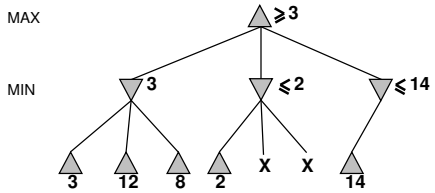
ESIMERKKI: ALFA–BEETA



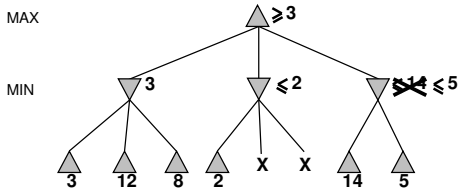
ESIMERKKI: ALFA–BEETA



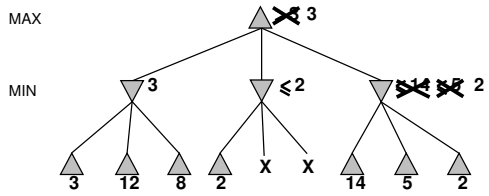
ESIMERKKI: ALFA–BEETA



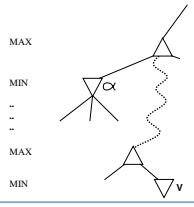
ESIMERKKI: ALFA–BEETA



ESIMERKKI: ALFA–BEETA



- Alfa–beeta-karsinnan nimi tulee ylläpidetyistä muuttujista:
  - $\alpha =$ parhaan (korkeimman) löydetyn valintamahdollisuuden arvo pelaajan MAX polulla
  - $\beta =$ parhaan (pienimmän) löydetyn valintamahdollisuuden arvo pelaajan MIN polulla
- Muuttujien α ja β arvoja päivitetään puuta läpikäytäessä ja heti kun jonkun solmun minimax-arvon voidaan päätellä jäävän heikommaksi kuin α tai β, jäljellä olevat haarat voidaan karsia.



- Alfa–beeta-karsinta ei vaikuta saatavan tuloksen arvoon.
- Karsinnan tehokkuus riippuu oleellisesti seuraajasolmujen tutkimisjärjestyksestä.
- Jos parhaat seuraajat pystyttäisiin tutkimaan ensin (mitä ei titenkään pystytä täydellisesti tekemään), putoaisi minimax-haun aikavaativuus $O(b^d)$ luokkaan $O(b^{d/2})$
- Efektiivinen haarautumisaste b putoaisi siis $\sqrt{b}$:hen. Esim. shakissa noin 35:stä noin 6:een. Hakupuuta voitaisiin tutkia suunnilleen kaksi kertaa niin pitkälle kuin pelkällä minimaxilla. Käytännössäkin päästään melko lähelle.
- Alfa–beeta-karsinta on yleisen *haaraudu ja rajoita* (branch and bound) -etsintäperiaatteen sovellus pelipuihin: Kun etsintä haarautuu, yritä laskea rajoja sille, kuinka hyviä tuloksia voi vielä ilmaantua, ja karsi etsintähaaroja näiden rajojen salliessa.

function ALPHA-BETA-DECISION(*state*) **returns** an action
  return the *a* in ACTIONS(*state*) maximizing MIN-VALUE(RESULT(*a*, *state*))

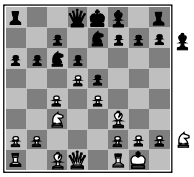
function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, α, β

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

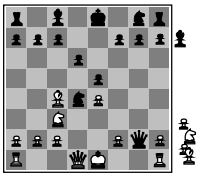
TOSIAIKAISET PÄÄTÖKSET

- Koska koko pelipuun läpikäyminen on oikeissa peleissä liian hidasta, pelipuun generoiminen on katkaistava johonkin syvyyteen ja arvioitava tilanteita heuristisesti sen mukaan kuinka hyviltä ne vaikuttavat
- Heuristisen funktion olisi arvotettava todelliset lopputilat oikeaan järjestykseen ja muut tilat sen mukaan kuinka suuret mahdollisuudet niistä on edetä voittoon.



Black to move

White slightly better



White to move

Black winning

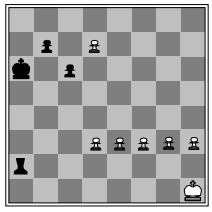
- Pelitilanteiden evaluointi ei saa olla liian hidasta.
- Heuristinen evaluointifunktio perustuu usein peliaseman *piirteiden* huomioimiseen.
- Usein kukin piirre f_i arvotetaan yksinään ja pelitilanteen s evaluoinniksi otetaan valittujen piirteiden lineaarikombinaatio

$$Eval(s) = \sum_{i=1}^{n} w_i f_i(s)$$

- Esimerkiksi shakissa piirteitä voisivat olla omien ja vastustajan sotilaiden, lähettien, tornien ja kuningattarien lukumäärien erotus.
- Näiden piirteiden painoja w_i puolestaan olisivat nappuloiden materiaaliset arvot (1,3,5 ja 9).

- Piirteiden lineaarikombinaatioon sisältyy ajatus piirteiden *riippumattomuudesta*.
- Kuitenkin esim. shakissa lähetin arvoon vaikuttaa esimerkiksi se onko toinenkin lähetti vielä mukana. Lisäksi lähetit ovat vahvoja loppupelissä, kun laudalla on tilaa.
- Siksi nykyiset shakkiohjelmat sisältävätkin myös piirteiden epälineaarisia kombinaatioita. Esimerkiksi lähettipari on painoarvoltaan enemmän kuin kaksi kertaa yhden lähetin paino, ja loppupelissä lähetin arvoa painotetaan enemmän kuin alussa.
- Shakissa eri piirteiden painoarvo saadaan satojen vuosien kokemuksen perusteella. Jos tällaista kokemusta ei ole, voidaan piirteiden painoarvot pyrkiä löytämään koneoppimisen avulla.
- Peliohjelman hyvyys riippuu merkittävästi evaluointifunktion laadusta.

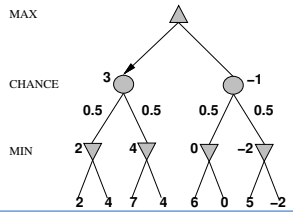
- Vielä on pohdittava sitä missä kohdassa haku katkaistaan.
- Yksinkertaisia katkaisuehtoja (cutoff test) saadaan esimerkiksi rajoittamalla haku tietylle kiinteälle syvyydelle tai syventämällä etsintää iteratiivisesti aikarajan puitteissa.
- Menetelmät kärsivät *horisonttiongelmasta* (horizon effect), joka syntyy kun pelitilanteen hyvyys muuttuu ratkaisevasti mutta vasta syvemmällä kuin mihin etsintä ehtii.



Black to move

SATTUMAN HUOMIOIMINEN

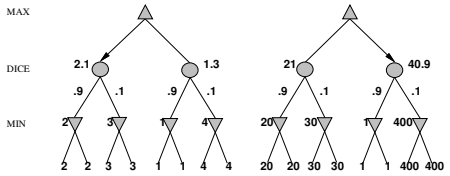
- Epädeterministisissä peleissä mukana satunnaisuutta kuten korttien sekoitus tai nopanheitto, esim. backgammon.
- Pelaaja ei tiedä vastustajan saamia (tai omia tulevia) silmälukuja eikä siis sallittuja siirtoja.
- Tavallista pelipuuta ei siis voi muodostaa, mutta pelipuuhun voidaan lisätä *sattumasolmuja* (chance nodes), joista johtavat kaaret nimetään satunnaisen elementin mahdollisilla tuloksilla ja niiden todennäköisyyksillä.



- Tarkkojen minimax-arvojen sijaan on tyydyttävä odotusarvon laskemiseen:

$$EMM(n) = \begin{cases} \text{UTILITY}(n) & \text{jos } n \text{ on loppusolmu} \\ \max_{s \in Succ(n)} EMM(s) & \text{jos } n \text{ on MAX-solmu} \\ \min_{s \in Succ(n)} EMM(s) & \text{jos } n \text{ on MIN-solmu} \\ \Sigma_{s \in Succ(n)} P(s) \cdot EMM(s) & \text{jos } n \text{ on sattumasolmu} \end{cases}$$

- Tilanteiden arvottaminen sattuman vallitessa on herkempi tehtävä kuin deterministisissä peleissä:



- Sattuman mukaantuominen nostaa pelipuun läpikäynnin vaativuuden luokkaan $O(b^m n^m)$, missä n on sattumasolmujen vaihtoehtojen maksimilukumäärä.
- Sattuman vallitessa pelipuita ei voi laskea pitkälle eteenpäin eikä karsintakaan ole helppoa koska tavallisesti sen avulla voidaan keskittyä todennäköisiin siirtojonoiin mutta satunnaisen elementin lisääminen rikkoo siirtojonojen todennäköisyyden.
- Jonkin verran karsintaa voidaan tehdä jos utiliteettifunktioiden arvoalueet ovat rajattuja.

NYKYISET PELIOHJELMAT

Shakki: IBM:llä kehitetty rinnakkaislaitteistoon perustuva Deep Blue voitti maailmanmestari Garry Kasparovin kuuden pelin ottelussa 1997. Hakunopeus keskimäärin 126 miljoonaa solmua sekunnissa. Normaali hakusyvyys 14 askelta. Perusmenetelmä iteratiivisesti syventävä α–β-haku, mutta kiinnostavimpia haaroja voitiin edetä jopa 40 askeleen (puolisiirron) syvyydelle. Evaluointifunktiossa 8000 piirrettä + laajat alku- ja loppupelikirjastot (5–6 nappulaa).

Tammi: Chinook maailmanmestariksi 1994 voittamalla (tosin luovutuksella) Marion Tinsleyn, joka oli hallinnut yli 40 vuotta. Ohjelma toimi PC:llä ja käytti α–β-karsintaa ja laski etukäteen kaikki 440 miljardia max. 8 nappulan loppupeliä.

Backgammon: Tesauron TD-GAMMON pelaajien Top 3:ssa. Neuraaliverkkoihin perustuva evaluointi ja haku 2–3 askelta.

Othello: Tietokoneet ylivoimaisia. (b luokkaa $5 - 15$)

Go: Ihmiset ylivoimaisia. ($b > 300$)