Kaheinimese mängud

- Mängupuu definitsioon

```
data Gtree a = Node a [Gtree a]
```

- Näide

```
Node 5 [Node 3 [Node 1 [],
                Node 2 []],
        Node 4 []]
```

- Kasulikke funktsioone

```
genGtree :: (a -> [a]) -> a -> Gtree a
genGtree f x = Node x [genGtree f t | t <- f x]

mapGtree :: (a -> b) -> Gtree a -> Gtree b
mapGtree f (Node x ts)
             = Node (f x) [mapGtree f t | t <- ts]
```

Kaheinimese mängud

- Mängupuu ehitamine

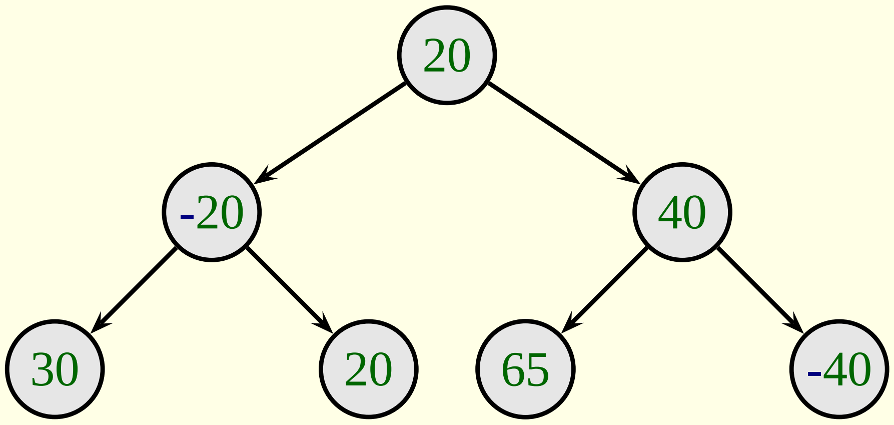
```
moves    :: Position -> [Position]

gametree :: Position -> Gtree Position
gametree p = genGtree moves p
```

- Puu kõrguse kärpimine

```
prune :: Int -> Gtree a -> Gtree a
prune 0     (Node x ts) = Node x []
prune (n+1) (Node x ts) = Node x [prune n t | t <- ts]
```

Seisu hindamine

- Iga seisu hinnang on numbriline väärtus;
- Kui ühe mäigija seisukohalt on hinnang w, siis vastasmängija seisukohalt on hinnang $-w$;
- Antud seisu hinnang võrdne selle lõppseisu hinnaguga milleni jõutakse, kui mõlemad mängijad teevad oma parimad käigud.
- Näide:



Seisu hindamine

- ”Minimax”-algoritm

```
minimax :: Gtree Int -> Int
minimax (Node x []) = x
minimax (Node x ts) = - minimum [minimax t | t <- ts]
```

- Seisu hinnangu leidmine

```
static   :: Position -> Int

evaluate :: Int -> Position -> Int
evaluate n = minimax . mapGtree static
                     . prune n . gametree
```

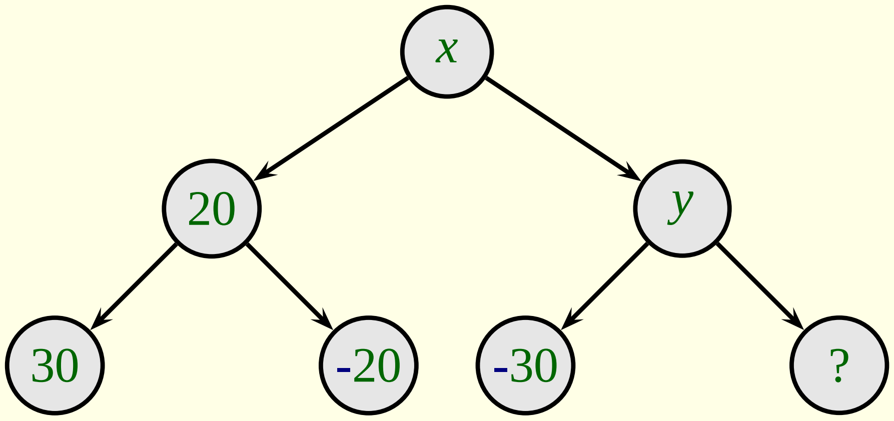
Käigu valimine

- Kui algseisu hinnang on w, siis valida tuleb käik mis viib seisuni hinnanguga $-w$.

```
makeMove :: Int -> Position -> Position
makeMove n p = chooseMin (head ps) (tail ps)
      where ps = [(p',evaluate n p') | p' <- moves p]

chooseMin (p,w) [] = p
chooseMin (p,w) ((p',w'):xs)
                   | w < w'    = chooseMin (p,w) xs
                   | otherwise = chooseMin (p',w') xs
```

Seisu hindamine

- Toodud algoritm pole optimaalne!
- Näide:



- Esitatud puus on $y \geq 30$ ja seega

```
x = - minimum [20,y] = -20
```

Seisu hindamine

- Uue algoritmi tuletamine

```
minimax = - minimum . mmx
mmx (Node x []) = [-x]
mmx (Node x ts) = map minimax ts
```

- Võrrandi mmx (2) teisendamine:

```
  map minimax ts
= map (-minimum . mmx) ts
= (map (-minimum) . map mmx) ts
= mapmin (map mmx ts)
    where mapmin = map (-minimum)
```

Seisu hindamine

- α/β-algoritm

```
minimax = - minimum . mmx
mmx (Node x []) = [-x]
mmx (Node x ts) = mapmin (map mmx ts)

mapmin (xs:xss) = -n : omit n xss
                  where n = minimum xs
omit n []       = []
omit n (xs:xss) | minleq n xs = omit n xss
                | otherwise   = -v : omit v xss
                  where v = minimum xs
minleq n []     = False
minleq n (x:xs) = x <= n || minleq n xs
```

Seisu hindamine

- Algoritm töötab efektiivsemalt siis, kui parimaid käike vaadeldakse esimestena

```
minimax = - minimum . mmx . bestfirst
bestfirst (Node x ts)
              = Node x (sortBy cmp (map bestfirst ts))
cmp (Node x _) (Node y _) = compare x y
```

- Seis võib hindamiseks olla liiga dünaamiline

```
dynamic :: Position -> Bool

prune 0 (Node x ts)
             | dynamic x = Node x [prune 0 t | t <- ts]
             | otherwise = Node x []
prune (n+1) (Node x ts) = Node x [prune n t | t <- ts]
```

Minimax moodul

```
module Minimax where

type Player = Bool

machine  = True
player   = False
opponent = not

class Position pos where
        moves   :: pos -> [pos]
        static  :: pos -> Int
        dynamic :: pos -> Bool
        win     :: Player -> pos -> Bool
```

Tikumäng

```
import Minimax

data Board = Board Player Int Int Int
initialBoard = Board player 3 5 7

movesP (Board pl x y z)
             =  [Board pl′ x′ y z | x′ <- [0..x-1]]
             ++ [Board pl′ x y′ z | y′ <- [0..y-1]]
             ++ [Board pl′ x y z′ | z′ <- [0..z-1]]
             where pl′ = opponent pl

winP pl′ (Board pl 0 0 0) = pl′ == pl
winP pl′ (Board pl x y z) = False
```

Tikumäng

```
staticP (Board pl x y z)
     = case ws of
         []    -> 1
         [x]   -> if x /= 1 then 1 else -1
         [x,y] -> if min x y == 1 || x /= y then 1 else -1
         _     -> 0
      where ws  = [i| i<-[x,y,z], i /= 0]

dynamicP (Board pl x y z) = x > 0 && y > 0 && z > 0

instance Position Board where
           moves   = movesP
           static  = staticP
           dynamic = dynamicP
           win     = winP
```