Male minimax

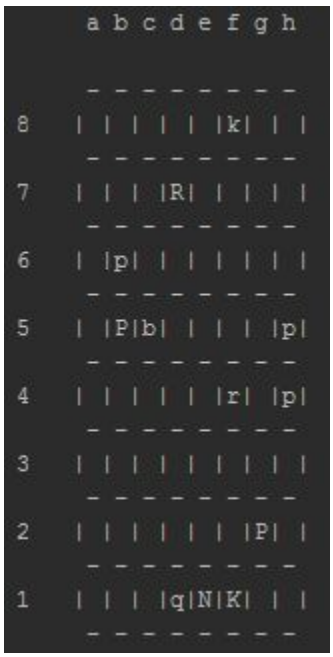
Töö eesmärgiks oli koostada algoritm, mis oskab mängida malet asjalikul tasemel. Selleks kasutasin otsingut graafil ja seeläbi lõin mängustrateegia, mis mind kui amatöör maletajat alati võidab. Programmi loomiseks kasutasin minimax'i ja alfa-beeta kärpimist, mille ideed sain aine praktikumide materjalidest 6 ja 7. Programmi kirjutasin ise keeles Scala ja mujalt koodi ei võtnud.

Kõige lihtsam on antud Scala projekt käivitada, kui laadida alla sbt aadressilt http://www.scala-sbt.org/download.html. Seejärel minna käsureal lahtipakitud programmi kausta ja anda käsk "sbt run". Teine võimalus on laadida alla IntelliJ IDEA, seal installeerida Scala plugin ja projekt avada antud IDE kaudu. Viimasel juhul tuleb käivitada fail Game.scala.

Programmi testimiseks saab valida failis Game.scala kes mängib valgete nuppudega ja kes mustadega. Näiteks kui valge mängija on "ManualControl" ja must mängija "AlphaBetaMinimax", siis saab mängida ise algoritmi vastu. See on üks hea viis testimiseks. Teine võimalus testimiseks on muuta valge mängija ka algoritmiks ja vaadata kuidas algoritmid üksteisega mängivad. Algselt on failis AlphaBetaMinimax.scala valitud sügavuseks muutujas "maxDepth" 3. Nii peab iga algoritmi käigu jaoks maksimaalselt 15 sekundit ootama. Kui see muuta 4 peale, siis muutub algoritm targemaks ja teeb loomulikemaid samme. Sel juhul tuleb iga käigu jaoks oodata kuni 1 minut, aga keskmiselt 10 sekundit.

Kui programm jookseb siis iga kägu järel prinditakse välja uus male laua seis ja laua äärtes on näha ka tavalist ruutude notatsiooni, a-h ja 1-8. Malendid on laual kirjas oma inglise keelsete tähistega: lipp - Q, kuningas - K, ettur - P, ratsu - N, vanker - R ja oda - B. Kusjuures valged nupud on tähistatud suurtähtedega ja mustad väiketähtedega. Kui mängida manuaalselt algoritmi vastu, siis tuleb sisestada iga käik algusruudu ja lõppruudu tähisega, näiteks kujul "b1c3". Kuninga poolse vangerdamise tarbeks on tähis "00" ja lipu poolse jaoks "000". Kui etturiga käiakse viimasele ruudule, siis saab kasutaja sisestada soovitud nupu tähise (q,r,b või n). Kui sisend antakse vale mingil mängu hetkel, siis öeldakse kasutajale erindiga, mis oli valesti ja kasutaja saab uuesti samast seisust jätkata.

Kui panna algoritm mängima teise identse algoritmiga, siis praeguse heuristikaga ja algoritmi sügavusega 4 lõppeb mäng alloleva seisuga, kus must mängja teeb valgele mati vankri ja oda abil. Kusjuures mõlemad mängijad on kaotanud küllaltki palju nuppe.



Tulemusega olen igati rahul, sest mul õnnestus luua küllaltki hea male algoritm. Loomulikult on veel mitmeid asju mida saaks algoritmi juures parandada. Üheks puuduseks on see, et praegu mängu käigus ei kontrollita, et vangerdamise algul ei oleks kuningas tule all, tegelikult sellises situatsioonis vangerdada ei tohiks. Samuti ei ole praegu võimalik teha *en Passant* käiku. (Kuna tegemist on väga harvade juhtumistega siis jätsin praegu need kõrvale.)

Veel ei kontrollita praegu seda, et mängija ei teeks käiku nii, et kuningas jääb käigu lõpus tule alla. Kui mängija sellise käigu teeb, siis algoritm võtab kuninga ära ja mäng saab läbi.

Algoritmi saaks muuta ka targemaks mitmel moel, näiteks saaks mängu alguses anda ette mingi tunnustatud male avang, või mõned avangud. Praegu käitub algoritm päris mängu alguses natuke iseäralikult. Lisaks nupu väärtustele on praegu algoritmile ette antud, et kui nuppe käia tagant ettepoole, siis see on hea. Üldjoontes peab selline idee paika, aga mitte alati. Saaks lisada mitmeid heuristilisi reegleid veel, nagu näiteks, et ratsu mängulaua serval ei ole väga hea jne.