TDT4136 Logic And Reasoning Systems

Exercise 8

Stian Hvatum (hvatum)
MTDT

24. januar 2012

Innhold

1 Ny algoritme

For å forbedre MinMaxPlayer, kopierte jeg Java-klassen til MinMaxImprovedPlayert.java. Her satt jeg opp max ply til 4, samt gjorde jeg slik at hvis ikke vi kunne vinne spillet, istedet for å returnere 0, returnerer jeg omvendt distanse til den andre spilleren. Dette gjør at vi "jakter" på den andre spilleren, inntil vi er så nære at vi kan kalkulere vinnertrekk med den "dumme" MinMax-algoritmen. Spilleren får nå litt delay pga. max ply er økt til 4, men på min maskin bruker MinMax-spilleren normalt mellom 3 og 10 sekunder på å gjøre et trekk. På tregere maskiner kan det være lurt å sette denne ned til 3.

I all hovedsak er det ply-dybden som gjør spilleren "smartere", mens distanse-måleren gjør ham aggressiv.

Forbedringen er gjort stort sett i metoden *evaluate*:

```
  public int evaluate(GameState state,int you){
    ...
    // Where are the other player at this move?
    int otherdriftx=otherp.x+otherp.dx*plydepth;
    int otherdrifty=otherp.y+otherp.dy*plydepth;
    int manhattan = Math.abs(otherdriftx-driftx) + Math.abs(otherdrifty-drifty);
    int maxManhattan = 40;
    return (maxManhattan-manhattan) * 10; // We want to get close
    }
```

2 Alpha-Beta

Alpha-Beta-algoritmen er implementert i AlphaBetaPlayer-klassen.

```
public ValuedMove createTree(GameState state, int d, int you, int alpha, int beta) {
        ... // Evaluate if we and that, just as in the original MinMax
            // then, inside the "for each possible move at this stage"
        for (int ddx = -1; ddx <= 1; ddx++) {
            for (int ddy = -1; ddy <= 1; ddy++) {
                ValuedMove v = createTree(control.createState(state, new Move(
                        ddx, ddy, state.playerturn)), d + 1, you, alpha, beta);
               ... finding min and max
                // this is my code for implementing AlphaBeta
                if (isMyTurn) {
                    if (max.val > alpha) {
                        alpha = max.val;
                    }
                    if (max.val >= beta) {
                        return max;
                    }
                } else {
                    if (min.val < beta) {
                        beta = min.val;
                    }
                    if (min.val <= alpha) {
                        return min;
                    }
                }
            }
        }
      ... //if no pruning, continue as before
    }
```

Nå som vi har implementert Alpha-Beta-algoritmen, kan vi sette opp max-ply til en del høyere. Jeg valgte også å legge til distanse-faktoren fra MinMaxImproved, da vi får flere forskjellige

verdier på nodene, og ikke bare 0 på alle som ikke er spesielt bra eller dårlige. Det gjør at vi kan skrelle mye mer.

3 Benchmark

Jeg testet litt hvor spreke de forskjellige algoritmene var, og kom fram til disse resultatene (alle er første skritt):

Algoritme:	Evalueringer	Tid(ca.)
Ply-dybde 3:		
MinMaxImproved:	174133	82 ms
AlphaBeta:	35743	46 ms
Ply-dybde 4:		
MinMaxImproved:	21560122	4937 ms
AlphaBeta:	298810	148 ms
Ply-dybde 5:		
MinMaxImproved:	1403584993	331998 ms
AlphaBeta:	5048358	2366 ms
Ply-dybde 6:		
MinMaxImproved:	-	- ms
AlphaBeta:	49620195	13623 ms
Ply-dybde 7:		
MinMaxImproved:	-	- ms
AlphaBeta:	998875588	245895 ms

Det er ganske tydelig at AlphaBeta er et bedre valg en MinMax når vi har mulighet til å implementere denne.