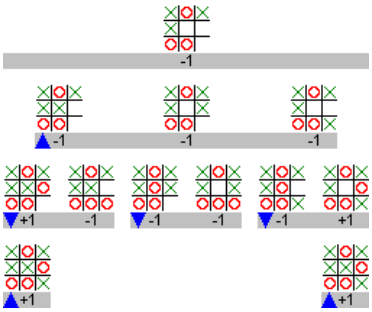
Games

Tic-Tac-Toe



backtracking algoritme met minimax strategie!

THRijswijk

1

minimax

```
int TicTacToe::chooseMove(Side s,
      int& bestRow, int& bestColumn) {
   Side opp(s==COMPUTER ?
                     HUMAN : COMPUTER);
   int value(s==COMPUTER ?
            HUMAN_WIN : COMPUTER_WIN);
   int simpleEval(positionValue());
   if (simpleEval!=UNCLEAR)
      return simpleEval;
   for (int r(0); r<board.numrows(); ++r)
      for (int c(0); c<board.numcols(); ++c)
         if (squareIsEmpty(r, c)) {
            place(r, c, s);
            int dc;
            int reply(chooseMove(opp, dc, dc));
            place(r, c, EMPTY);
            if (s==COMPUTER && reply>value ||
                s==HUMAN && reply<value) {
               value=reply;
               bestRow=r;
               bestColumn=c;
            }
         }
   return value;
}
```

THRijswijk

2

TTT

aantal aanroepen chooseMove bij eerste zet (computer begint)

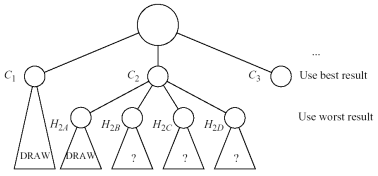
- Maximaal: 1+9+9x8+9x8x7+ ... + 9x8x7x6x5x4x3x2x1 = 986410
- Stoppen als er een winnaar is = 549946
- Toepassen alpha-beta pruning = 18297
- Toepassen transposition table = 7954
- Zoek identieke stellingen (draaien en spiegelen) = ???

THRijswijk

3

Games

Alpha-Beta pruning



Zie p. 398.
After H_{2A} is evaluated, C_2, which is the minimum of the H_2's, is at best a draw. Consequently, it cannot be an improvement over C_1. We therefore do not need to evaluate H_{2B}, H_{2C}, and H_{2D}, and can proceed directly to C_3.

THRijswijk

4

alpha-beta

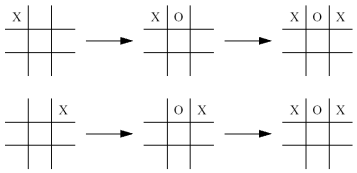
```
int TicTacToe::chooseMove(Side s,
      int& bestRow, int& bestColumn,
      int alpha, int beta) {
   Side opp(s==COMPUTER ?
                     HUMAN : COMPUTER);
   int value(s==COMPUTER ? alpha : beta);
   int simpleEval(positionValue());
   if (simpleEval!=UNCLEAR) return simpleEval;
   for (int r(0); r<board.numrows(); ++r)
      for (int c(0); c<board.numcols(); ++c)
         if (squareIsEmpty(r, c)) {
            place(r, c, s); int dc;
            int reply(chooseMove(opp, dc, dc,
                                 alpha, beta));
            place(r, c, EMPTY);
            if (s==COMPUTER && reply>value ||
                s==HUMAN && reply<value) {
               value=reply;
               if (s==COMPUTER) alpha=value;
               else beta=value;
               bestRow=r; bestColumn=c;
               if (alpha>=beta) return value;
            }
         }
   return value;
}
```

THRijswijk

5

Games

Transpostions



Zie p. 400.
Two searches that arrive at identical positions.

THRijswijk

6

transpositions

```
class Position {
public:
   Position(const matrix<int>& theBoard):
         board(theBoard) {
   }
   bool operator<(const Position& rhs) const;
private:
   matrix<int> board;
};

bool Position::operator<(const Position & rhs)
      const {
   for (int i(0); i<board.numrows(); ++i)
      for (int j(0); j<board.numcols(); ++j)
         if (board[i][j]!=rhs.board[i][j])
            return board[i][j]<rhs.board[i][j];
   return false;
}

class TicTacToe {
//...
private:
   map<Position, int> transpositions;
};
```

THRijswijk

7

transpositions

```
int TicTacToe::chooseMove(Side s,
      int& bestRow, int& bestColumn,
      int alpha, int beta, int depth) {
   Position thisPosition(board);
   if (depth>=3 && depth<=5) {
      MapItr itr(transpositions.find(thisPosition));
      if (itr!=transpositions.end())
         return (*itr).second;
   }
// idem ...
            int reply(chooseMove(opp, dc, dc,
                                 alpha, beta, depth+1));
// idem ...
            if (alpha>=beta) goto Done;
         }
      }
Done:
   if (depth>=3 && depth<=5)
      transpositions[thisPosition]=value;
   return value;
}
```

THRijswijk

8