Does Queen + King = Checkmate ?

Supercomputing Challenge

Final Report

April 6, 2011

Team 014

Aspen Elementary

Alex Ionkov and Donald Poston

Teacher: Zeynep Unal

Summary

Our project was based on game theory. People have been trying to make chess programs since computers were invented but it was a challenge to make such a program. Famous computer scientists like Robert Wiener and Alan Turing made the first attempts to solve the problem. Los Alamos scientists created the first program to play chess in 1956. In 1981 the first computer program has made a US masters rating. After 2000 computers are strong enough to beat the worlds best grandmasters rated 2700 and above.

We chose the queen and king vs king because we knew how to do the king-queen checkmate in chess and thought it would be better to start out easy. Our mentor recommended that we start out with less pieces because it was our first year of the Supercomputing Challenge.

Our main goals are: to learn how to write computer programs, and learn how to write a computer programs that play chess.

Our main accomplishments are: we learned how to use NetLogo to write computer programs, we learned how to create a chess board and chess pieces, we learned how to write report procedures that return all the possible moves for a piece, and we learned how to create procedures that find the best move black or white using the minimax algorithm.

We had issues like: NetLogo was to slow to play against the program with the decision tree deeper than 3 levels with depth less than or equal to three, it was easy to get a draw due to three-fold repetition or 50 moves.

Problem Statement

We are trying to write a program that would find the best move for a player with a king and a queen against an opponent with a single king.

Method

The program has to follow the rules of chess [1].

The only way black can win is by check-mating. The best thing that can happen to white is if black draws white by either losing it's queen, 3-fold-repetition (when the exact same position occurs 3 times), if black and white have each made 50 moves, or if black stalemates (white has no moves at all and is not in check).

We are trying to write a program that would find the best move for a player with a king and a queen against an opponent with a single king. In order to find the best move we create a tree that starts from the current position and goes to all possible positions to a certain number of moves. Then we evauluate all positions and choose the best one.

We used the minimax algorithm to find the best move. This is the pseudocode for the algorithm:

```
to report best-move [ position depth]
    ifelse is-checkmate? or is-stalemate? or depth < = 0 [
        report eval-position
    ]
    [
        let moves calculate-all-moves
        let val -1
        foreach moves [
            move-piece ?
            val = max val -(best-move depth -1)
            move-piece-back ?
        ]
        report val
    ]
end
```

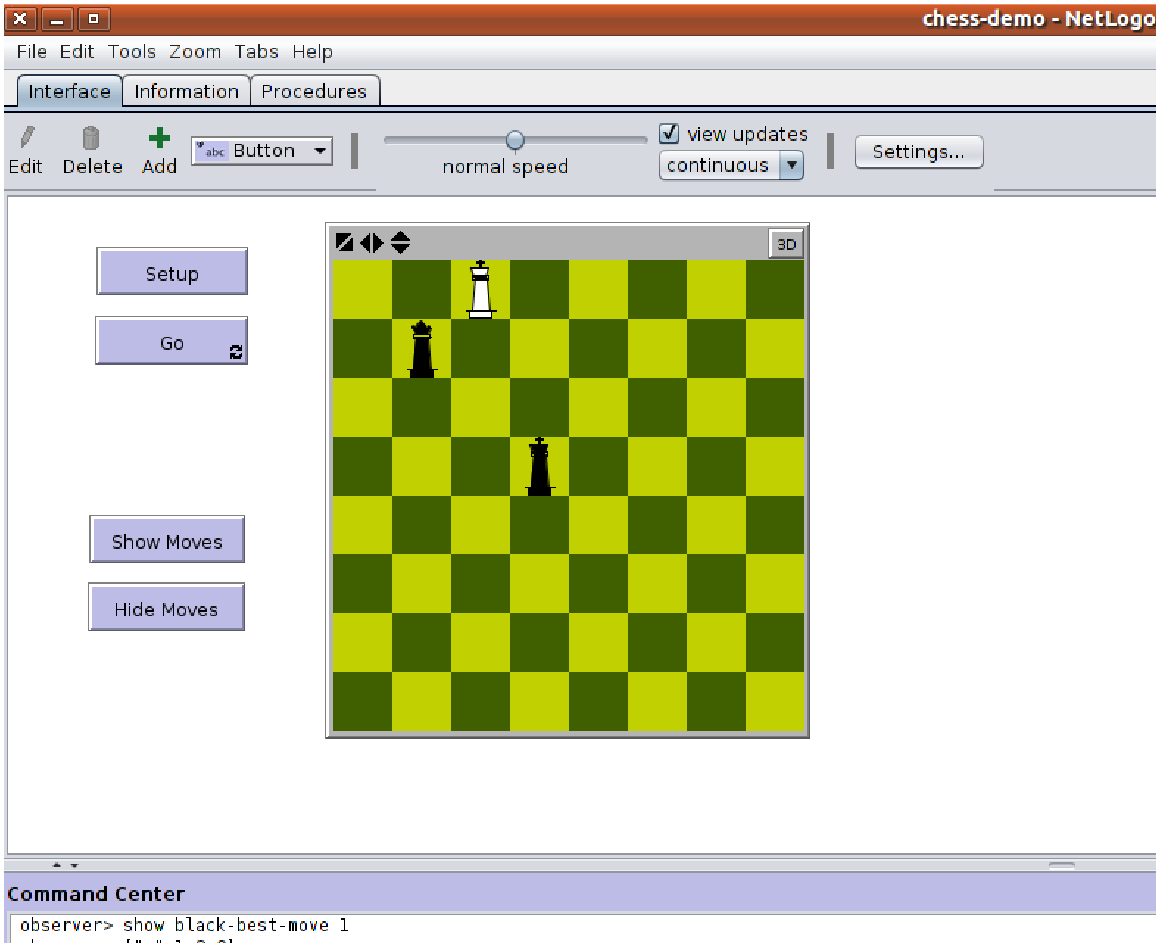
We wanted to use the alpha-beta algorithm[3][4], but we didn't have time to. We used the following table to evaluate the position.

Stalemate	1
Check	-0.2
Check-mate	-1
Black queen taken	1

Accomplishments

We used NetLogo for our project, we created a model that has 64 patches that look like a chess board. The model has three agents: black queen, black king, and white king. The human can use the mouse to move the white king and the computer moves the black queen and king.

Depth	**Time (seconds)**
0	0.01
2	0.73
4	70.91
6	1181.13



```
observer> show black-best-move 1
```

This is a screenshot of our program interface.

Conclusion:

We learned how to use some advanced programming concepts like lists of values, recursion, and procedures.

Our choices of algorithms and programming language weren't very appropriate for the problem we picked. We are planning to try to use better algorithms (like alpha-beta search) and a programming language that is faster like Python or Go. If we do programming again we might use Python because it will be easier to do a more complicated problem and because we could add more decision tree levels.

References:

1. **United States Chess Federation's Official Rules of Chess**, Fifth Edition

2. C. Shannon **Programming a Computer For Playing Chess**, National IRE Convention, March 1949

3. R. Feldmann **Computer Chess: Algorithms and Heuristics for a Deep Look into the Future**, Proc. of the 24th Seminar on Current Trends in Theory and Practice of Informatics (SOFSEM-97) LNCS 1338, 511-522, Springer Verlag.

4. D.E. Knuth, R.W. Moore **An Analysis of Alpha – Beta Prunning, Artifical Intelligence**, 6, pp 293-326, 1975

Program Code:

```
breed [blackqueens blackqueen]
breed [blackkings blackking]
breed [whitekings whiteking]
breed [grayqueens grayqueen]
breed [graykings grayking]

to setup
  clear-all
  create-blackqueens 1 [
    set color black
    set shape "chess queen"
  ]
  create-blackkings 1 [
    set color black
    set shape "chess king"
  ]

  create-whitekings 1 [
    set color white
    set shape "chess king"
  ]
  ask patches [
    ifelse  (pxcor + pycor) mod 2 = 0
    [
      set pcolor [ 64 96 0 ]
    ]
    [
      set pcolor [ 192 208 0]
    ]
  ]
  scatter-pieces
  if check-end? whiteking-moves [
    stop
  ]
end

to scatter-pieces
  loop [
    ask turtles
    [
      set ycor random 8
      set xcor random 8
    ]

    let dostop true
    ; check if kings are next to each other
    ask blackkings[
      if count whitekings-on neighbors = 1 [
        set dostop false
      ]
    ]

    ; check if two pieces on same square
    ask turtles [
      if any? other turtles-here[
        set dostop false
      ]
    ]

    if dostop [
      stop
    ]
  ]
end

to-report blackking-moves
  let moves []
  ask blackkings [
    ask neighbors [
      if count whitekings-on neighbors = 0 [
        set moves lput (list pxcor pycor) moves
      ]
    ]
  ]
  report moves
end

to-report whiteking-moves
    let moves []
    let x 0
    let y 0

    ask whitekings [
      set x xcor
      set y ycor
      ask neighbors [
        if count blackkings-on neighbors = 0 [
          set moves lput (list pxcor pycor) moves
        ]
      ]
    ]

    ask whitekings [
      die
    ]
    let qmoves blackqueen-moves false
    create-whitekings 1 [
      set color white
      set shape "chess king"
      setxy x y
    ]

    foreach moves [
      if member? ? qmoves [
        set moves remove ? moves
      ]
    ]

    report moves
end

to-report line-moves [L x0 y0 tx ty can-take]
  let x x0 - tx
  let y y0 - ty
  while [ x >= 0 and y >= 0 and x < 8 and y < 8] [
    ifelse can-take
    [
      ifelse (any? blackkings-on (patch x y)) [
        set x -1
      ]
      [
        set L lput (list x y) L
        set x x - tx
        set y y - ty
      ]
    ]
    [
      ifelse (any? turtles-on (patch x y)) [
        set x -1
      ]
      [
        set L lput (list x y) L
        set x x - tx
        set y y - ty
      ]
    ]

  ]
  report L
end

to-report blackqueen-moves [ can-take ]
    let moves []

    ask blackqueens [
      set moves line-moves moves xcor ycor 1 0 can-take
      set moves line-moves moves xcor ycor -1 0 can-take
      set moves line-moves moves xcor ycor 0 1 can-take
      set moves line-moves moves xcor ycor 0 -1 can-take
      set moves line-moves moves xcor ycor 1 1 can-take
      set moves line-moves moves xcor ycor -1 -1 can-take
      set moves line-moves moves xcor ycor -1 1 can-take
      set moves line-moves moves xcor ycor 1 -1 can-take
    ]

    report moves
end

to go
    if mouse-down? [
      let x round mouse-xcor
      let y round mouse-ycor
      let moves whiteking-moves
      if member? (list x y) moves [
        ask whitekings [
          setxy x y
        ]

        ask patch x y [
          if any? blackqueens-here [
            ask blackqueens [
              die
            ]
          ]
        ]

        if check-end? moves [
          stop
        ]

        move-black
      ]
    ]
end

to-report check-end? [ moves ]
  if empty? moves [
    ifelse is-check?
      [
        user-message "checkmate"
      ]
      [
        user-message "stalemate"
      ]

    report true
  ]

  if count blackqueens = 0 [
    user-message "draw"
    report true
  ]

  report false
end

to-report is-check?
  let qmoves blackqueen-moves true
  let bkpos []
  ask whitekings [
    set bkpos (list xcor ycor)
  ]

  let qpos []
  ask blackqueens [
    set qpos (list xcor ycor)
  ]

  report member? bkpos qmoves
end

to move-black
  reset-timer
  let move black-best-move 6

  show timer
  let piece (item 0 move)
  let x (item 1 move)
  let y (item 2 move)

  ifelse piece = "k"
  [
    ask blackkings [
      setxy x y
    ]
  ]
  [
    ask blackqueens [
      setxy x y
    ]
  ]
end

to show-white-moves
  let qmoves blackqueen-moves false
  let kmoves blackking-moves

  foreach qmoves [
    let x (item 0 ?)
    let y (item 1 ?)
    ask patch x y [
      if not any? turtles-here [
        sprout-grayqueens 1 [
          set color [64 64 64]
          set shape "chess queen"
        ]
      ]
    ]
  ]

  foreach kmoves [
    let x (item 0 ?)
    let y (item 1 ?)
    ask patch x y [
      if not any? turtles-here [
        sprout-graykings 1 [
          set color [64 64 64]
          set shape "chess king"
        ]
      ]
    ]
  ]
end

to hide-white-moves
  ask graykings [
    die
  ]

  ask grayqueens [
    die
  ]
end

to-report white-best-move [ level ]
 let best-value -1
 let best-x -1
 let best-y -1
 let best-piece ""
 foreach whiteking-moves [
   let x (item 0 ?)
   let y (item 1 ?)
   let oldx 0
   let oldy 0
   ask whitekings [
     set oldx xcor
     set oldy ycor
     setxy x y
   ]

   let queen-taken false
   ask patch x y [
     if any? blackqueens-here [
       set queen-taken true
       ask blackqueens [
         die
       ]
     ]
   ]

   let val -1
   ifelse level = 0 [
     set val eval-position
   ]
   [
     set val (item 3 (black-best-move (level - 1)))
   ]

   if val > best-value [
     set best-value val
     set best-x x
     set best-y y
   ]

   ask whitekings [
     setxy oldx oldy
   ]

   if queen-taken [
     create-blackqueens 1 [
       set color black
       set shape "chess queen"
       setxy x y
     ]
   ]
 ]

 report (list "k" best-x best-y best-value)
end

to-report black-best-move [ level ]
 let best-value 1
 let best-x -1
 let best-y -1
 let best-piece ""
 foreach black-moves [
   let piece (item 0 ?)
   let x (item 1 ?)
   let y (item 2 ?)
   let oldx 0
   let oldy 0

   ifelse piece = "k"
   [
     ask blackkings [
       set oldx xcor
       set oldy ycor
       setxy x y
     ]
   ]
   [
     ask blackqueens [
       set oldx xcor
       set oldy ycor
       setxy x y
     ]
   ]

   let val -1
   ifelse level = 0 [
     set val eval-position
   ]
   [
     set val (item 3 (white-best-move (level - 1)))
   ]

   if val < best-value [
     set best-value val
     set best-x x
     set best-y y
     set best-piece piece
   ]

   ifelse piece = "k"
   [
     ask blackkings [
       setxy oldx oldy
     ]
   ]
   [
     ask blackqueens [
       setxy oldx oldy
     ]
   ]
 ]

 report (list best-piece best-x best-y best-value)
end

to-report black-moves
  let L []
  let qmoves blackqueen-moves false
  foreach qmoves [
    set L lput (list "q" ( item 0 ?) ( item 1 ?)) L
  ]
  let kmoves blackking-moves
  foreach kmoves [
    set L lput (list "k" (item 0 ? ) ( item 1 ?)) L
  ]

  report L
end

to-report eval-position
  if is-stalemate? [
    report 1.0
  ]

  if is-check? [
    report -0.2
  ]
  if is-checkmate?[
    report -1.0
  ]
  if count blackqueens = 0 [
    report 1.0
  ]

  report 0.0
end

to-report is-stalemate?
  if is-check? [ report false]
  if length whiteking-moves = 0 [ report true]
  report false
end

to-report is-checkmate?
  if is-check? and length whiteking-moves = 0 [ report true]
  report false
end
```