Quick start

go run tic-tac-toe-4107056019.go

Use a state structure to represent a tree node

```go
type state struct {
    graph []int8
    val   int8
    alpha int8
    beta  int8
}

func newState() state {
    return state{
        alpha: -128,
        beta:  127,
    }
}
```

Use a slice `t` to represent the current state. I map 9 cells to a 1D slice

| 0 | 1 | 2 |

| 3 | 4 | 5 |

| 6 | 7 | 8 |

Use a Boolean variable `halt` to represent whether we should stop searching, i.e., whether we reach the goal state.

```go
func main() {
    // 0  : empty
    // -1 : O AI
    // 1  : X human

    t := []int8{0, 0, 0, 0, 0, 0, 0, 0, 0}
    halt := false
    round := 0
    rand.Seed(time.Now().UnixNano())
```

Loop if we have not yet reached the goal state (line37). Use showTicTacToe() to print current game board (line39). If the round is even, switch to human, and otherwise, switch to computer. (line40 and line57)

In the human round, get the number typed by the real player (line41-47). If the real player input 0 exit game. (line48-51) If the real player types an invalid number, the program will notice her or him retype again (line52-line55). Get next best state by calling minimax() (line61).

Before calling minimax(), store the current state (line59-60). We can compare the new state with the old one to get the changed cell(line62-68). Give the next best state to current state `t` (line69)

```
37      for !halt {
38          fmt.Printf( format: "Round %d\n", round)
39          showTicTacToe(t)
40          if round&1 == 0 { // human round
41              var where int
42              fmt.Printf( format: "Your move: ")
43              _, err := fmt.Scanf( format: "%d\n", &where)
44              for err != nil {
45                  fmt.Fprintf(os.Stderr,  format: "input error, %v try again:", err)
46                  _, err = fmt.Scanf( format: "%d", &where)
47              }
48              if(where == 0){
49                  fmt.Println( a...: "You give up. The program is terminated")
50                  os.Exit( code: 0)
51              }
52              for where < 1 || where > 9 || t[where-1] != 0 {
53                  fmt.Fprintf(os.Stderr,  format: "input error, try again:")
54                  _, err = fmt.Scanf( format: "%d", &where)
55              }
56              t[where-1] = 1
57          } else { // AI round
58              // find the best step
59              current := newState()
60              current.graph = t
61              next := minimax(current, min)
62              var move int
63              for i, v := range next.graph {
64                  if v != t[i] {
65                      move = i
66                      break
67                  }
68              }
69              t = next.graph
70              fmt.Println( a...: "AI's move:", move+1)
71          }
```

Check if we reach the goal state after playing (line76). If we reach the goal state, show the board and the result (line77), and then set` the Boolean variable `halt` to be true (line85). Otherwise, jump to line37 unless reaching the goal state.

```
72
73          fmt.Println()
74          round++
75
76          if done, res := goalTest(t); done {
77              showTicTacToe(t)
78              if res < 0 {
79                  fmt.Println( a...: "AI win, 回家再練10年吧")
80              } else if res == 0 {
81                  fmt.Println( a...: "tie, 讓你一把")
82              } else {
83                  fmt.Println( a...: "AI 是不會輸的 ^_^")
84              }
85              halt = true
86          }
87      }
88  }
```

How to show tic-tac-toe board? Use showTicTacToe()

```
90  func showTicTacToe(this []int8) {
91      for k, v := range this {
92          switch v {
93          case 0:
94              fmt.Printf( format: "|%d", k+1)
95          case 1:
96              fmt.Printf( format: "|X")
97          case -1:
98              fmt.Printf( format: "|O")
99          }
100         if k%3 == 2 {
101             fmt.Println( a...: "|")
102         }
103     }
104 }
```

How to get the best next state?
My strategy is to fill one of all empty cells in 1 (X) or -1 (O) sequentially until reaching the goal state. When reaching the goal state, return the value (-1 means O wins, 1 means X wins, and 0 means tie) to the previous state, i.e., I use depth-first search to traversal the minimax tree.

Check if the cell is empty or not (0 is empty)(line111). If it is empty cell, fill the cell in 1 or -1 (line112-124). Test if reaching the goal state (line127). If it has reached the goal state, store the result to the `child.val` (line133). Otherwise, recursively get all possible states, i.e., generate child nodes (line131).
The child node (child state) Inherits the alpha value and beta value from its parent node (parent state) (line128, 129).

```
106 func minimax(parent state, minMax bool) state {
107     var ret state
108     initRet := true
109     for i, v := range parent.graph {
110         // Generate all possible chlidren nodes
111         if v == 0 {
112             child := newState()
113             child.graph = make([]int8, len(parent.graph))
114             // copy
115             for i, v := range parent.graph {
116                 child.graph[i] = v
117             }
118
119             // max -> fill with 1
120             // min -> fill with -1
121             if minMax == max {
122                 child.graph[i] = 1
123             } else {
124                 child.graph[i] = -1
125             }
126
127             done, val := goalTest(child.graph)
128             child.alpha = parent.alpha // copy alpha value to child's alpha
129             child.beta = parent.beta   // copy beta value to child's beta
130             if !done {
131                 child.val = minimax(child, !minMax).val
132             } else {
133                 child.val = val
134             }
```

If we fill the cell for the first time, initialize `ret` by child state and consider `ret` as a threshold (line136-137).

Update the parent node's value by comparing two children nodes' value. i.e., return the values of children nodes to its parent node. (line143 – 151).

If the values of two children states are 0 simultaneously, randomly pick one. (line153-155). I have already set the seed at line35.

Do alpha-beta pruning (line159-163).

```
136             if initRet {
137                 ret = child
138                 initRet = false
139             }
140
141             // min level 更新 parent beta
142             // max level 更新 parent alpha
143             if minMax == min && child.val < ret.val {
144                 ret = child
145                 parent.beta = child.val
146             }
147
148             if minMax == max && child.val > ret.val {
149                 ret = child
150                 parent.alpha = child.val
151             }
152
153             if child.val == ret.val && rand.Float32() > .5 {
154                 ret = child
155             }
156
157             // min level 的 val 值若比 min 的 alpha 小則忽略其他節點
158             // max level 的 val 值若比 max 的 beta 大則忽略其他節點
159             if minMax == min && child.val < parent.alpha {
160                 break
161             }
162             if minMax == max && child.val > parent.beta {
163                 break
164             }
165         }
166     }
167     parent.val = ret.val
168
169     return ret
170 }
```

GoalTest()

Declare 2D integer slice to store 8 goal states (line185-194), and use an inner product to test the goal state (line 196-203).

If all the inner product values are not 3 or -3 (line204), it means two conditions, a tie or just not done yet. Multiply all values in the slice (line205-208), and if the result is 0, we can infer that there are 0 in some cells. It means that the state is just dot done. (line 210). Otherwise, the state is a tie (line212).

```
180 // @return[0] if the state is at destination
181 // @return[1] -1 when AI `O` win
182 //             1 when human `X` win
183 //             0 when tie
184 func goalTest(state []int8) (bool, int8) {
185     winState := [][]int8{
186         {1, 1, 1, 0, 0, 0, 0, 0, 0},
187         {0, 0, 0, 1, 1, 1, 0, 0, 0},
188         {0, 0, 0, 0, 0, 0, 1, 1, 1},
189         {1, 0, 0, 1, 0, 0, 1, 0, 0},
190         {0, 1, 0, 0, 1, 0, 0, 1, 0},
191         {0, 0, 1, 0, 0, 1, 0, 0, 1},
192         {1, 0, 0, 0, 1, 0, 0, 0, 1},
193         {0, 0, 1, 0, 1, 0, 1, 0, 0},
194     }
195
196     for _, oneOfGoalState := range winState {
197         res := innerProduct(oneOfGoalState, state)
198         if res == -3 {
199             return true, -3
200         } else if res == 3 {
201             return true, 3
202         }
203     }
204
205     var product int8 = 1
206     for _, k := range state {
207         product *= k
208     }
209     if product == 0 { // have not done
210         return false, 0
211     }
212     return true, 0 // tie
213 }
```

Implement an inner product function.

```
172  func innerProduct(a []int8, b []int8) int8 {
173      var sum int8 = 0
174      for i := range a {
175          sum += a[i] * b[i]
176      }
177      return sum
178  }
```